Αριστοτελειο Πανεπιστημιο Θεσσαλονικης

Πολυτεχνικη Σχολη
Τμημα Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Ηλεκτρονικης και Υπολογιστων
Εργαστηριο Επεξεργασιας Πληροφοριων και Υπολογισμων

Διπλωματικη Εργασια

Υπολογισμός Σημασιολογικής Ομοιότητας Έργων Λογισμικού χρησιμοποιώντας Σχόλια

Ιωάννης Λόϊας (ΑΕΜ 8183)

Επιβλέπων καθηγητής:
Ανδρέας Λ. Συμεωνίδης
Συνεπίβλεψη:
Υπ. Δρ. Εμμανουήλ Κρασανάκης

24 Φεβρουαρίου 2019

Περίληψη

Το διαδίκτυο άλλαξε ριζικά τα μέσα και την ταχύτητα της μετάδοσης πληροφοριών και έχει δημιουργήσει νέες προοπτικές στον τομέα της ανάπτυξης κώδικα. Μια από αυτές είναι η ανάπτυξη αποθετηρίων κώδικα, τα οποία φιλοξενούν μεγάλο αριθμό έργων λογισμικού που μπορούν να χρησιμοποιηθούν από τους προγραμματιστές. Ωστόσο, οι προγραμματιστές καταχρώνται συχνά τον πηγαίο κώδικα που αντλούν από τέτοια αποθετήρια χωρίς να αποδίδουν τον αρχικό δημιουργό.

Η αντιγραφή πηγαίου κώδικα με τη μορφή λογοκλοπής οδηγεί σε μείωση της αναζήτησης εναλλακτικών λύσεων για προβλήματα που λύνονται από υπάρχουσες μεθόδους και δεν ανταμείβει τους προγραμματιστές που ανέπτυξαν τον αρχικό κώδικα. Για να αντιμετωπιστούν αυτά τα φαινόμενα, είναι απαραίτητο να αναπτυχθούν κατάλληλα συστήματα ανίχνευσης λογοκλοπής, τα οποία υπολογίζουν την ομοιότητα μεταξύ έργων λογισμικού. Οι τρέχουσες απόπειρες ανάπτυξης τέτοιων συστημάτων τείνουν να εξετάζουν μόνο τον πηγαίο κώδικα των έργων λογισμικού, είτε αντιμετωπίζοντας το περιεχόμενό του ως αδόμητο κείμενο είτε αναλύοντας τις σχέσεις οντοτήτων πηγαίου κώδικα χρησιμοποιώντας αφηρημένα συντακτικά δέντρα ή δέντρα κλήσης συναρτήσεων. Αυτές οι πρακτικές δεν συνυπολογίζουν τα συνοδευτικά σχόλια, τα οποία ενδεχομένως περιέχουν σημαντικές πληροφορίες για τον κώδικα που περιγράφουν. Εμείς υποθέτουμε ότι τέτοια σχόλια μπορούν να συμβάλουν στη βελτίωση της κατανόησης του πότε τα δύο έργα λογισμικού είναι παρόμοια.

Σε αυτή την εργασία αναπτύσσουμε ένα σύστημα που μπορεί να ανακαλύψει ομοιότητες πηγαίου κώδικα από μια πολύπλευρη οπτική που αναλύει τη σημασιολογία και τη δομή τόσο του πηγαίου κώδικα όσο και των συσχετιζόμενων σχολίων. Το σύστημα αυτό χρησιμοποιεί μια πληθώρα διαδεδομένων τεχνικών επεξεργασίας πηγαίου κώδικα για να συγκρίνει τα έργα λογισμικού και να παράγει τιμές ομοιότητας για διαφορετικά χαρακτηριστικά. Για παράδειγμα, υποστηρίζει αλγορίθμους διανυσματοποίησης, όπως η επεξεργασία λέξεων του πηγαίου κώδικα μέσω της σακούλας λέξεων και των χαρακτηριστικών tf-idf, με σκοπό την εφαρμογή μεθόδων σύγκρισης διανυσμάτων σε επίπεδο αρχείου και συνάρτησης. Εναλλακτικά, μπορεί να χρησιμοποιήσει τη λανθάσουσα σημασιολογική ανάλυση (LSA), η οποία μειώνει τον θόρυβο που προκύπτει από τη χρήση διαφορετικών όρων με την ίδια σημασιολογία. Το σύστημα εφαρμόζει επίσης έναν αριθμό υφιστάμενων και καινοτόμων μεθόδων που βασίζονται σε γράφους για να συγκρίνουν δέντρα κλήσης συναρτήσεων. Τέλος, όλες οι μέθοδοι είναι σε θέση να συμπεριλάβουν τα σχόλια του πηγαίου κώδικα κατά τον υπολογισμό της ομοιότητας μεταξύ έργων λογισμικού.

Δοκιμάσαμε το σύστημά μας σε δύο σύνολα δεδομένων σχεδίων λογισμικού που εξήχθησαν από το GitHub: ένα σύνολο δεδομένων που περιέχει έργα λογισμικού και τις διακλαδώσεις τους (forks) που σχετίζονται με τη λέξη-κλειδί "Pacman" και ένα σύνολο που περιέχει έργα λογισμικού διάφορων τομέων με τις διακλαδώσεις τους. Από τα πειράματά μας προέκυψε ότι η χρήση σχολίων πηγαίου κώδικα βοηθά να εντοπίσουμε αν δύο έργα του συνόλου δεδομένων "Pacman" είναι διακλαδώσεις του ίδιου αρχικού έργου, παράγοντας πιο περιγραφικές μετρήσεις για την αξιολόγηση από αλγόριθμους που δεν τα χρησιμοποιούν. Στη συνέχεια, για το δεύτερο σύνολο δεδομένων χρησιμοποιήσαμε την έκδοση των μεθόδων που λαμβάνουν υπόψη τα σχόλια του πηγαίου κώδικα και έχουν υψηλή διακριτική ικανότητα στο προηγούμενο σύνολο για να ανακαλύψουμε ποια έργα λογισμικού ήταν διακλαδώσεις του ίδιου έργου. Όλοι οι αλγόριθμοι βρέθηκαν να είναι περίπου ίσοι για να αποδώσουν υψηλότερες ομοιότητες στις διακλαδώσεις των ίδιων έργων. Ωστόσο, οι αλγόριθμοι που συγκρίνουν τα δέντρα κλήσης συναρτήσεων ήταν πιο αξιόπιστοι στην ανακάλυψη παρόμοιων έργων, αποδίδοντας σχεδόν μηδενικές ομοιότητες κατά τη σύγκριση των διακλαδώσεων ανόμοιων έργων.

ARISTOTLE UNIVERSITY OF THESSALONIKI

FACULTY OF ENGINEERING
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
DEPARTMENT OF COMPUTERS & ELECTRONICS
INFORMATION PROCESSING LABORATORY

DIPLOMA THESIS

Measuring Semantic Similarity of Software Projects Using Comments

Ioannis Loias (AEM 8183)

Supervisor:
Andreas L. Symeonidis
Co-supervisor:
PhD Cand. Emmanouil Krasanakis

Abstract

The internet has radically changed the means and speed of information transmission. In the field of code development, it has created new prospects by creating software repositories, which host a large number of software projects that can be used by developers. However, developers often plagiarize source code that they have not developed without attributing the original creator.

Copying source code in the form of plagiarism leads to less search for alternative solutions to existing methods and does not reward developers who developed the original code. For these reasons, it is necessary to develop appropriate systems that can detect plagiarism by measuring the similarity between software projects. Current approaches tend to only consider the source code of these projects, either treating its content as unstructured text or analyzing source code entity relations using abstract syntax trees or call trees. These practices donnot account for accompanying comments, which possibly contain important information about the code they describe. We theorize that such comments can help improve understanding of when two projects are the same.

In this work we develop a system that can discover source code similarities from a multifaceted perspective that analyzes the semantics and structure of both the source code and its related comments. The system we develop employs a multitude of widespread source code processing techniques to compare software projects and produce similarity values for different features. For example, it supports vectoring algorithms, such as word processing of source code through bag-of-words and tf-idf features, for the purpose of applying vector comparison methods at the file and function levels. The outcome of vectoring can also be refined with Latent Semantic Analysis (LSA) that reduces the noise resulting from the use of different terms with the same semantics. Our system also implements a number of existing and novel graph-based methods to compare function call trees. Finally, all methods are able to account for source code comments when calculating similarities between software projects.

We tested our system on two datasets of software projects extracted from GitHub; one comprised of software projects and their forks related to the keyword 'Pacman' and one comprised of software projects and their forks across different domains. We asserted that employing source code comments helps better detect whether two projects of the 'Pacman' dataset are forks of the same project, producing more descriptive evaluation results than algorithms that do not use them. Then, for the second dataset, we used the version of our methods that account for source code comments to help discover which software projects were forks of the same one in this dataset. All algorithms were found to be approximately equal in yielding higher similarities for forks of the same projects. However, algorithms that compare function call trees could more reliably discover similar projects and yielded almost zero similarities when comparing forks of dissimilar ones.

Ευχαριστίες

Στο σημείο αυτό, θα ήθελα να ευχαριστήσω τους ανθρώπους που διαδραμάτισαν καθοριστικό ρόλο στην εκπόνηση της παρούσας διπλωματικής εργασίας. Αρχικά, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή αυτής της εργασίας, τον κύριο Ανδρέα Συμεωνίδη, για την εμπιστοσύνη που μου επέδειξε με την ανάθεση της εργασίας, τις στοχευμένες παρατηρήσεις και την άψογη συνεργασία που είχαμε καθ' όλη τη διάρκεια της. Επίσης, θα ήθελα να ευχαριστήσω ιδιαίτερα τον υποψήφιο διδάκτορα, Εμμανουήλ Κρασανάκη, για την ενθάρρυνση, την καθοδήγηση, την άμεση ανταπόκριση του σε οποιαδήποτε ανάγκη προέκυπτε καθώς και για την τεχνική του βοήθεια με παροχές βιβλιοθηκών που προσαρμόστηκαν στο σύστημα που υλοποιήθηκε στα πλαίσια αυτής της διπλωματικής εργασίας. Τέλος, δε θα μπορούσα να παραλείψω τους γονείς μου και τις αδερφές μου, για την ενθάρρυνση και τη στήριξη που μου προσφέρουν όλα αυτά τα χρόνια.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κατάλογος Αλγορίθμων

1 Εισαγωγή

1.1 Γενικά

Από τα τέλη του 20ου αιώνα πραγματοποιείται σταδιακή μετάβαση από την παραδοσιακή βιομηχανία σε μια οικονομία που βασίζεται στο χειρισμό των πληροφοριών, σημάνοντας την άφιξη της εποχής της πληροφορίας [1]. Σε αυτή τη μετάβαση παίζει σημαντικό ρόλο ο Παγκόσμιος Ιστός, ο οποίος αναπτύχθηκε από τον Βρετανό Tim Berners Lee στον Ευρωπαϊκο Οργανισμό Πυρηνικών Ερευνών (CERN).

Με τη ραγδαία αύξηση τους κόστους ανάπτυξης λογισμικού, άρχισε να αναπτύσσεται και μια αντίστοιχη βιομηχανία λογισμικού. Από την άλλη, οι κατασκευαστές εξοπλισμού διένειμαν δωρεάν το λογισμικό, συμπεριλαμβάνοντας το κόστος του στο κόστος του εξοπλισμού. Λόγω του ανταγωνισμού αυτού, η βιομηχανία λογισμικού άρχισε να λαμβάνει τεχνικά μέτρα, όπως τη διανομή μόνο δυαδικών αντιγράφων για τα προγράμματα υπολογιστών, με σκοπό να αποτρέψει τους χρήστες από την μελέτη και τροποποίηση του λογισμικού. Το 1980 η νομοθεσία περί πνευματικής ιδιοκτησίας επεκτάθηκε και στα προγράμματα υπολογιστών. Ως απάντηση σε αυτές τις πρακτικές, το 1985 ιδρύθηκε το Ίδρυμα Ελεύθερου Λογισμικού (Free Software Foundation-FSF) από τον Richard Stallman, ο οποίος ανέπτυξε έναν ορισμό για το ελεύθερο λογισμικό με σκοπό την προάσπιση της ελευθερίας των ανθρώπων να χρησιμοποιούν, μελετούν, τροποποιούν, αντιγράφουν και βελτιώνουν ένα πρόγραμμα [2,3]. Αργότερα, υπήρξαν κι άλλες παρόμοιες δράσεις, όπως η ίδρυση της Πρωτοβουλίας Ανοιχτού Κώδικα (Open Source Initiative - OSI) από τους Eric Raymond και Bruce Perens.

Οι δράσεις αυτές, σε συνδυασμό με την εξέλιξη του παγκόσμιου ιστού, αποτέλεσαν θεμέλιο λίθο για τη δημιουργία αποθηκών λογισμικού, στις οποίες οι προγραμματιστές μπορούν μεταξύ άλλων να αναζητούν έργα λογισμικού τρίτων, καθώς και για την δημιουργία υπηρεσιών ερώτησης-απάντησης μεταξύ προγραμματιστών. Χαρακτηριστικό παράδειγμα αποθήκης λογισμικού είναι το GitHub [4–7], το οποίο παρέχει υπηρεσίες φιλοξενίας λογισμικού μέσω διαδικτύου και ελέγχου της διαδικασίας ανάπτυξης. Η αποθήκη αυτή χρησιμοποιείται κυρίως για τη διαχείριση κατανεμημένων εκδόσεων πηγαίου κώδικα, παρέχει έλεγχο πρόσβασης και λειτουργίες συνεργασίας όπως παρακολούθηση σφαλμάτων, αιτήματα χαρακτηριστικών και διαχείριση εργασιών για κάθε έργο. Παρόμοια υπηρεσία είναι και το Sourceforge [8], το οποίο βασίζεται στο διαδίκτυο για να προσφέρει στους προγραμματιστές μια κεντρική ηλεκτρονική τοποθεσία ελέγχου και διαχείρισης έργων λογισμικού ανοιχτού κώδικα. Για παράδειγμα, παρέχει χώρο αποθήκευσης πηγαίου κώδικα, παρακολούθηση σφαλμάτων, λίστες αλληλογραφίας προγραμματιστών και χρηστών, φόρουμ υποστήριξης χρηστών, σχόλια και αξιολογήσεις από τους χρήστες, δελτίο ενημερώσεων που αφορούν το έργο λογισμικού και μικρο-blog για δημοσίευση ενημερώσεων έργου. Τέλος, ένας πολύ γνωστός ιστότοπος ερωτήσεων-απαντήσεων για προγραμματιστές είναι το StackOverflow [9–11]. Ο ιστότοπος αυτός χρησιμεύει ως πλατφόρμα για τους χρήστες να ρωτούν και να απαντούν σε ερωτήσεις και μέσω της ενεργού συμμετοχής τους, να ψηφίζουν θετικά είτε αρνητικά αυτές και να τις επεξεργάζονται. Έτσι, φιλοξενούνται λύσεις σε κοινά προβλήματα του προγραμματισμού οι οποίες χαρακτηρίζονται απο αξιοπιστία λόγω της αλληλεπίδρασης των χρηστών.

Η δημιουργία πλατφόρμων όπως οι παραπάνω άλλαξε ριζικά τα δεδομένα στην ανάπτυξη λογισμικού. Υπάρχει πλέον διαθέσιμος τεράστιος όγκος προγραμμάτων στις αποθήκες αυτές. Έτσι, ο προγραμματιστής έχει στη διάθεσή του διάφορα μέσα από τα οποία μπορεί να αντλήσει πηγαίο κώδικα που υλοποιεί τη λειτουργικότητα ή μέρη της λειτουργικότητας που καλείται κάθε φορά να αναπτύξει. Από την ευκολία αυτή όμως πηγάζει το πρόβλημα της λογοκλοπής, η οποία ορίζεται από τον S. Hannabuss [12] ως "η πράξη της μίμησης ή της αντιγραφής ή της χρήσης κάποιας άλλης δημιουργίας ή ιδέας χωρίς άδεια και την παρουσίασή της ως δική της". Στο πεδίο της τεχνολογίας λογισμικού, ως λογοκλοπή θεωρείται η μη πίστωση το αρχικού συγγραφέα. Για παράδειγμα, ένας προγραμματιστής μπορεί να χρησιμοποιεί τον πηγαίο κώδικα που ανέπτυξε κάποιος άλλος χωρίς να αναφέρει το όνομα του τελευταίου. Η παρούσα εργασία μελετά αυτού του είδους τη λογοπκλοπή και όχι την αντιγραφή του αλγορίθμου ή του προγράμματος που παράγεται.

Η λογοκλοπή πηγαίου κώδικα έχει εμφανιστεί εδώ και χρόνια. Ένας χώρος που συναντάται συχνά είναι αυτός της εκπαίδευσης. Μια έρευνα του 2002 έδειξε ότι το 85,4% σε μια τάξη 137 φοιτητών στο Πανεπιστήμιο Monash και το 69,3% σε μια τάξη 150 φοιτητών στο Πανεπιστήμιο Swinburne συμμετείχε σε λογοκλοπή του πηγαίου κώδικα [13]. Ο κύριος λόγος που συμβαίνει αυτό είναι η έλλειψη χρόνου από την πλευρά των διδασκόντων για τον ενδελεχή έλεγχο των προγραμμάτων καθώς επίσης και η έλλειψη συστημάτων που υλοποιούν συγκρίσεις σε εύλογο χρονικό διάστημα.

Παρομοίως, είναι συχνό το φαινόμενο σε έργα ανοικτού κώδικα να χρησιμοποιούνται τμήματα άλλων έργων. Για παράδειγμα, σχετικές έρευνες [14, 15] δείχνουν ότι περισσότερο από το 50% των αρχείων πηγαίου κώδικα επαναχρησιμοποιούνται σε περισσότερα από ένα έργα ανοιχτού κώδικα.

1.2 Ορισμός του Προβλήματος

Οι προγραμματιστές χρησιμοποιούν τον πηγαίο κώδικα κάποιου άλλου για διάφορους λόγους. Για παράδειγμα, είναι πιθανότερο να έχουν διορθωθεί τυχόν αστοχίες σε πηγαίο κώδικα που έχει δοκιμαστεί και ελεγχθεί σε πραγματικά συστήματα. Έτσι, αυτός αναμένεται να είναι πιο φερέγγυος από κώδικα που συγγράφεται εκ νέου. Επίσης, η χρήση έτοιμων λύσεων μειώνει δραστικά το χρόνο που απαιτείται για να υλοποιηθούν νέα συστήματα. Ωστόσο, όταν οι προγραμματιστές χρησιμοποιούν έτοιμο κώδικα άλλων δεν αντιμετωπίζουν το πρόβλημα που καλούνται να λύσουν με κριτική σκέψη. Έτσι, στερούνται της ικανότητας να παράγουν καινοτομίες ή να αναζητούν καλύτερες λύσεις για το πρόβλημα με το οποίο ασχολούνται. Ειδικά σε επίπεδο εκπαίδευσης, αυτό θεωρείται και ακαδημαϊκή ατιμωρησία.

Για το λόγο αυτό, είναι σημαντική η ανάπτυξη συστημάτων υπολογισμού ομοιότητας πηγαίου κώδικα. Τέτοια συστήματα προσπαθούν να ανακαλύψουν την ομοιότητα περιεχομένου μεταξύ του πηγαίου κώδικα διαφορετικών έργων λογισμικού. Για παράδειγμα, αν δύο αρχεία πηγαίου κώδικα έχουν παρόμοιους όρους με παρόμοιες συχνότητες τότε αυτά θα μπορούσαν να θεωρηθούν όμοια [16–18]. Άλλα συστήματα βασίζονται στα δέντρα κλήσης συναρτήσεων που εξάγονται από τον κώδικα [19]. Τέτοια συστήματα θεωρούν ότι ο τρόπος που οι συναρτήσεις καλούν η μία την άλλη είναι αντιπροσωπευτικός του εκάστοτε πηγαίου κώδικα και η ομοιότητα μπορεί να υπολογιστεί βάσει αυτού. Τέλος, κάποια συστήματα [20–23] βασίζονται στην ομοιότητα αφηρημένων συντακτικών δέντρων του κώδικα.

Ένα σύνηθες μειονέκτημα των παραπάνω συστημάτων είναι ότι δε λαμβάνουν υπόψη τα σχόλια του πηγαίου κώδικα, τα οποία και θεωρούν περιττή πληροφορία. Επίσης, το γεγονός ότι υποστηρίζουν μία μόνο μέθοδο υπολογισμού ομοιότητας δεν παρέχει συνολική εικόνα της λογοκλοπής και μπορεί να οδηγήσει σε εσφαλμένα αποτελέσματα. Για παράδειγμα, στην περίπτωση που ο πηγαίος κώδικας εξετάζεται ως σακούλα λέξεων, χάνεται δομική πληροφορία, ενώ όταν εξετάζεται αποκλειστικά η μορφολογία ενός δέντρου κλήσης δεν αξιοποιείται η πληροφορία των όρων που χρησιμοποιούνται. Σε κάθε περίπτωση, δύο προγράμματα μπορούν να έχουν παρόμοια σημασιολογία ή δομή ενώ υλοποιούν τελείως διαφορετικά προβλήματα. Τα συστήματα αυτά, επομένως, δεν εξετάζουν σφαιρικά τον πηγαίο κώδικα. Κάτι τέτοιο θα μπορούσε να γίνει αν για τη διεξαγωγή του αποτελέσματος αξιολογούνταν ο κώδικας από όσο το δυνατόν περισσότερες διαφορετικές οπτικές. Αυτό συμβαίνει στην περίπτωση που εξετάζεται τόσο η δομή όσο και το περιεχόμενό του. Σε αυτήν την εργασία αναπτύσσουμε ένα σύστημα το οποίο προσπαθεί να καλύψει τα κενά αυτά.

1.3 Σκοπός της διπλωματικής εργασίας

Σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάλυση βασικών εννοιών που σχετίζονται με τη σύγκριση και την ανάλυση του πηγαίου κώδικα από τη σκοπιά των γράφων κλήσεων συναρτήσεων. Πραγματοποιείται επίσης μια μελέτη γύρω από τη χρηστικότητα των σχολίων του πηγαίου κώδικα για τη βελτίωση της αναγνώρισης λογοκλοπών. Γενικά, οι μεταβολές που παρατηρούνται μεταξύ παρόμοιων έργων λογισμικού δεν χρειάζονται εξειδικευμένες γνώσεις προγραμματισμού για να εντοπιστούν. Οι αλλαγές αυτές γίνονται κυρίως σε επίπεδο όρων όπως μεταβολή του ονόματος μιας μεταβλητής ή αλλαγή στη σειρά των δηλώσεων μεταβλητών ή κάποιων πράξεων που δεν έχουν επίπτωση στο πρόγραμμα. Έτσι, ο τρόπος κλήσης των συναρτήσεων παραμένει σχεδόν ανεπηρέαστος. Από την άλλη, τα σχόλια, επειδή περιγράφουν τη διαδικασία της μεθόδου που συνοδεύουν θεωρείται ότι παρέχουν πληροφορία που μπορεί να βοηθήσει στον υπολογισμό ομοιότητας.

Επίσης, περιγράφουμε τα συστήματα της σχετικής βιβλιογραφίας που έχουν υλοποιηθεί μέχρις στιγμής και παρέχουμε μια βασική περιγραφή των κοινών διαδικασιών. Στα πλαίσια αυτής της ανάλυσης, περιγράφουμε μεθόδους που χρησιμοποιούνται για τη σύγκριση πηγαίου κώδικα, όπως η λανθάνουσα σημασιολογική ανάλυση, η αναπαράσταση του κώδικα ως σακούλα λέξεων και μεθόδους που βασίζονται σε αφηρημένα συντακτικά δέντρα και δέντρα κλήσης.

Τέλος, αναπτύχθηκε ένα νέο σύστημα το οποίο υπολογίζει την ομοιότητα μεταξύ των προγραμμάτων με μεθόδους τόσο της υπάρχουσας βιβλιογραφίας όσο και με νέες. Αυτό το σύστημα είναι κατάλληλο για χρήση από μηχανικούς λογισμικού για την αναγνώριση όμοιων έργων λογισμικού που φαινομενι-

κά δείχνουν διαφορετικά. Μπορεί επίσης να εφαρμοστεί και από εκπαιδευτικούς, όπου από παρόμοια κατατεθημένα έργα λογισμικού μπορούν να αναγνωρίσουν ύποπτα για περαιτέρω διερεύνηση.

Το παραπάνω σύστημα πραγματοποιεί πολυεπίπεδη ανάλυση του κώδικα και δεν περιορίζεται στην ανάλυση μόνο του περιεχομένου ή μόνο της δομής του. Για την ακρίβεια, υπολογίζονται διάφορες τιμές ομοιότητας με βάση το περιεχόμενο, τη δομή και τα σχόλια του πηγαίου κώδικα. Ο χρήστης εισάγει ένα σύνολο έργων λογισμικού τα οποία συγκρίνονται μεταξύ τους και τα αποτελέσματα ομοιότητας που εξάγονται είναι προσαρμοσμένα στις μεταξύ τους διαφορές.

Μια σημαντική δυνατότητα του εργαλείου που αναπτύχθηκε είναι ότι μπορεί να αναγνωρίσει πότε ένα αρχείο πηγαίου κώδικα αποτελεί απόσπασμα ενός ευρύτερου αρχείου. Ακόμη, ενημερώνει το χρήστη για το αν η ομοιότητα συναντάται σε επίπεδο περιεχομένου ή δομής. Τέλος, η ομοιότητα του πηγαίου κώδικα παρουσιάζεται με εύληπτο τρόπο, χρησιμοποιώντας διαγράμματα που επιτρέπουν την άμεση εξαγωγή συμπερασμάτων για τις συγκρίσεις που διηξήχθησαν. Όλα τα παραπάνω πραγματοποιούνται σε ικανοποιητικό χρόνο. Με βάση τα κριτήρια που θέτουμε, το σύστημα βελτιώνει τις λύσεις που χρησιμοποιούν μόνο μία μέθοδο σύγκρισης.

1.4 Διάρθρωση του κειμένου

Στο *Κεφάλαιο 1* εισαγάγαμε τον αναγνώστη στο αντικείμενο της τρέχουσας διπλωματικής και ορίσαμε συνοπτικά το πρόβλημα που καλούμαστε να αντιμετωπίσουμε. Στη συνέχεια, στο *Κεφάλαιο 2*, περιγράφουμε τις μεθόδους υπολογισμού ομοιότητας που χρησιμοποιούνται στη βιβλιογραφία για τον πηγαίο κώδικα. Στο *Κεφάλαιο 3* αναλύουμε συστήματα που έχουν επιχειρήσει να λύσουν παρόμοια προβλήματα. Αναφερόμαστε και πάλι σε ορισμένα συστήματα, αναλύοντάς όμως περισσότερο τα τεχνικά τους χαρακτηριστικά.

Στη συνέχεια, στο *Κεφάλαιο 4* αναλύουμε το σύστημα που υλοποιήσαμε. Αναφερόμαστε αναλυτικά σε όλα τα υποσυστήματα και τα εργαλεία που το συγκροτούν και περιγράφουμε τη λειτουργία τους. Το σύστημα αυτό αξιολογείται στο *Κεφάλαιο 5*, όπου παρουσιάζεται αναλυτικά η διαδικασία και τα αποτελέσματα αξιολόγησης των διάφορων μεθόδων σε μια πληθώρα έργων λογισμικού.

Στο *Κεφάλαιο 6* περιγράφουμε την υπηρεσία ιστού που χρησιμοποιήθηκε ως διεπαφή για το σύστημά μας. Αναφέρουμε, επίσης, μερικά παραδείγματα, ώστε να γίνει κατονοητή η σημασία των γραφημάτων που εξάγονται από αυτό.

Τέλος, στο *Κεφάλαιο 7* παρουσιάζουμε συμπεράσματα που προκύπτουν από το σύστημά μας σχετικά με την επίλυση του προβλήματος που τέθηκε. Επιπλέον, αναφέρουμε ανοιχτά θέματα και πιθανές βελτιώσεις ή επεκτάσεις που θα μπορούσαν να πραγματοποιηθούν.

2 Υπόβαθρο

Τα συστήματα που εμφανίζονται στη βιβλιογραφία επεξεργάζονται τον πηγαίο κώδικα με διαφορετικούς τρόπους. Άλλα τον αντιμετωπίζουν ως κείμενο και άλλα χρησιμοποιούν μια δομημένη αναπαράστασή του στην οποία εφαρμόσουν κάποιον αλγόριθμο σύγκρισης. Παρά τις διαφορές που μπορεί να εμφανίζονται στις επιμέρους μεθόδους, αναγνωρίζουμε τέσσερις βασικές κατηγορίες ανάλυσης και σύγκρισης κώδικα: τη λανθάνουσα σημασιολογική ανάλυση, την επεξεργασία ως κείμενο, την αναπαράσταση με σύμβολα και την αναπαράσταση σε μορφή δέντρων.

2.1 Λανθάνουσα σημασιολογική ανάλυση

Η λανθάνουσα σημασιολογική ανάλυση (Latent Semantic Analysis - LSA) είναι μια τεχνική επεξεργασίας φυσικής γλώσσας που πραγματοποιεί ανάλυση σχέσεων μεταξύ συνόλων αρχείων με βάση τους όρους τους οποίους περιέχουν. Η τεχνική αυτή περιλαμβάνει αλγόριθμους που εφαρμόζονται σε συλλογές αρχείων κειμένου και πηγαίου κώδικα. Για την εφαρμογή της σε μια τέτοια συλλογή απαιτείται αρχικά κατάλληλη λεξικογραφική προεπεξεργασία (βλπ. Αλόριθμος 1) και στη συνέχεια αναπαράσταση της συλλογής σε έναν πίνακα $A : m \times n$ μεταξύ n αρχείων και m όρων που βρίσκονται σε αυτά [17, 18, 24].

Αλγόριθμος 1 Προεπεξεργασία αρχείων για καλύτερη εφαρμογή λανθάνουσας σημασιολογικής ανάλυσης

1. Αφαίρεση σχολίων
2. Διαχωρισμός ενωμένων λέξεων (π.χ. CamelCase → Camel Case)
3. Μετατροπή κεφαλαίων σε πεζά
4. Διαγραφή των όρων χωρίς σημασιολογικό περιεχόμενο (stop words). Τέτοιοι όροι μπορεί να είναι λέξεις-κλειδιά γλωσσών προγραμματισμού, παράμετροι του αρχείου (π.χ. το όνομα του προγραμματιστή), αριθμητικοί χαρακτήρες, σύμβολα και λέξεις που αποτελούνται από μόνο ένα γράμμα και λέξεις που εμφανίζονται σε όλα τα αρχεία πηγαίου κώδικα.
5. Αφαίρεση ορών που εμφανίζονται σε ένα μόνο αρχείο
6. Αφαίρεση επιθημάτων από λέξεις

Τα στοιχεία A_{ij} του πίνακα A περιέχουν τη συχνότητα με την οποία εμφανίζεται ο όρος i στο αρχείο j. Πρακτικά, κάθε αρχείο αντιστοιχεί σε διάνυσμα όρων. Με τον τρόπο αυτό κάθε αρχείο μπορεί να αναπαρασταθεί σε έναν διανυσματικό χώρο ώστε να είναι δυνατή η σύγκριση των διανυσμάτων και κατ' επέκταση των αρχείων. Συνήθως οι όροι του διανύσματος είναι σταθμισμένοι ανάλογα με το μέγεθος κάθε αρχείου και τη γενική σημαντικότητα τους. Τα διανύσματα όρων έχουν μη μηδενικές τιμές στις θέσεις που αντιστοιχούν λέξεις υψηλής σημασίας για το κάθε αρχείο ενώ σε κάθε άλλη περίπτωση μηδενικές τιμές. Για παράδειγμα, λέξεις που βρίσκονται σε πολλά αρχεία με μικρή συχνότητα στο καθένα θα έχουν μικρότερη τιμή από λέξεις που βρίσκονται σε λιγότερα αρχεία με μεγαλύτερη συχνότητα στο καθένα. Με την εφαρμογή της διαδικασίας αυτής, το διάνυσμα αντιπροσωπεύει με μεγαλύτερη ακρίβεια το αρχείο πηγαίου κώδικα στο διανυσματικό χώρο. Συνεπώς, ο υπολογισμός της ομοιότητας μεταξύ δύο αρχείων θα είναι πιο αξιόπιστος.

Κάθε στοιχείο του πίνακα όρων-αρχείων μπορεί να αναπαρασταθεί ως γινόμενο τριών παραγόντων:

$$A_{ij} = f_{ij} g_i n_j \tag{2.1α'}$$

όπου f_{ij} είναι ο αριθμός εμφανίσεων του όρου i στο αρχείο j, g_i είναι το γενικό βάρος του όρου i και n_j είναι ο παράγοντας κανονικοποίησης με βάση το μέγεθος του αρχείου [25]. Συνήθως οι ποσότητες αυτές υπολογίζονται ως:

$$g_i = \frac{1}{\sqrt{\sum_i f_{ij}^2}} \tag{2.1β'}$$

$$n_j = \Big(\sum_i (g_i f_{ij})^2 \Big)^{-\frac{1}{2}} \tag{2.1γ'}$$

Αφότου εφαρμοστούν τα βάρη αυτά ώστε να παραχθεί ο σταθισμένος πίνακας A, εφαρμόζεται ο αλγόριθμος ανάλυσης σε ιδιάζουσες τιμές (Singular Value Decomposition-SVD) ώστε να αποσυνθέσει τον σταθμισμένο πίνακα σε ξεχωριστούς πίνακες [26]. Οι πίνακες αυτοί καταγράφουν την ομοιότητα μεταξύ

των όρων και μεταξύ των αρχείων πηγαίου κώδικα σε διάφορες διαστάσεις στο χώρο. Στόχος είναι να αναπαρασταθούν οι σχέσεις μεταξύ των όρων σε μειωμένο χώρο διαστάσεων. Με αυτόν τον τρόπο, ο θόρυβος (δηλαδή η ποικιλία των λέξεων που χρησιμοποιούνται για να περιγράψουν την ίδια έννοια) αφαιρείται από τα δεδομένα και ανακαλύπτονται οι σημαντικές σχέσεις μεταξύ όρων και αρχείων [27]. Η λανθάνουσα σημασιολογική ανάλυση στοχεύει στο να βρει υποκείμενες (λανθάνουσες) σχέσεις μεταξύ διαφορετικών όρων που έχουν την ίδια σημασία αλλά ποτέ δεν εμφανίζονται στο ίδιο αρχείο [28]. Έτσι, η χρήση αυτού του αλγορίθμου αποσκοπεί στον υπολογισμό της ομοιότητας των αρχείων χωρίς να επηρεάζεται από διαφορετικούς όρους με την ίδια σημασιολογία.

Ο πίνακας όρων αρχείων μπορεί να διασπαστεί (αποσυντεθεί) σε τρεις πίνακες χρησιμοποιώντας τις ιδιοτιμές και τα ιδιοδιανύσματα του αρχικού πίνακα. Η αποσύνθεση παράγει έναν $m \times r$ πίνακα όρων-διαστάσεων U, ένα $r \times r$ μοναδιαίο πίνακα Σ και έναν $n \times r$ πίνακα αρχείων-διαστάσεων V. Ο αρχικός πίνακας μπορεί να ανασκευαστεί από το γινόμενο των τριών πινάκων $U\Sigma V^T$, όπου V^T είναι ο ανάστροφος του πίνακα αρχείων-διαστάσεων. Ο βαθμός r του αρχικού πίνακα είναι ο αριθμός των μη-μηδενικών στοιχείων της διαγωνίου του. Η ανάλυση σε ιδιάζουσες τιμές μπορεί να παρέχει μια προσέγγιση βαθμού k του αρχικού πίνακα, όπου το k αντιπροσωπεύει των αριθμό των διαστάσεων που επιλέγονται και ισχύει ότι $k \leq r$. Αυτή η διαδικασία είναι γνωστή ως μείωση των διαστάσεων και περιλαμβάνει την περικοπή τριών πινάκων. Συγκεκριμένα, οι πρώτες k στήλες των τριών πινάκων που προέκυψαν κατά την αποσύνθεση του αρχικού πίνακα παραμένουν άθικτες. Οι υπόλοιπες είτε διαγράφονται είτε μηδενίζονται. Οι αντίστοιχοι πίνακες που προκύπτουν από τη μείωση των διαστάσεων σε k είναι οι $U_k : m \times k$, $\Sigma_k : k \times k$ και $V_k : n \times k$. Χρησιμοποιώντας αυτούς τους πίνακες πραγματοποιείται η προσέγγιση του αρχικού πίνακα με ανάλυση σε ιδιάζουσες τιμές βαθμού k:

$$A_k = U_k \Sigma_k V_k^T \tag{2.2α'}$$

Αν ο πίνακας A είναι ένας πίνακας όρων-αρχείων, το γινόμενο των πινάκων $X = A^T A$ είναι ένας πίνακας αρχείων-αρχείων. Έτσι το κελί x_{ij} υποδηλώνει την ομοιότητα του αρχείου i με το αρχείο j. Στην περίπτωση που εφαρμοστεί η ανάλυση σε ιδιάζουσες τιμές ο πίνακας αρχείων-αρχείων προκύπτει από το γινόμενο:

$$similarities = (V_k \Sigma_k)(V_k \Sigma_k)^T \tag{2.2β'}$$

2.2 Υπολογισμός Ομοιότητας με χρήση συμβόλων

Μια δεύτερη κατηγορία συστημάτων εύρεσης ομοιότητας πηγαίου κώδικα είναι αυτά που ακολουθούν τη διαδικασία αντικατάστασης του πηγαίου κώδικα με σύμβολα (tokens) [29, 30]. Σκοπός αυτής της πρακτικής είναι ο υπολογισμός της ομοιότητας να μην επηρεάζεται από αλλαγές της δομής ενός προγράμματος ή από μετονομασία συναρτήσεων και μεταβλητών με την ίδια σημασιολογική ερμηνεία. Πρόκειται για μια διαδικασία όπου ο κώδικας διασπάται σε κομμάτια όπως λέξεις-κλειδιά ή φράσεις. Κάθε κομμάτι κώδικα αναπαρίσταται με ένα σύμβολο. Τα σύμβολα που χρησιμοποιούνται επιλέγονται από ένα ενιαίο σύνολο συμβόλων βάσει της σημασιολογίας των εκφράσεων που αντικαθιστούν. Στη συνέχεια, η ομοιότητα δύο προγραμμάτων υπολογίζεται συγκρίνοντας τα σύνολα των συμβόλων μεταξύ τους, όπως δημιουργήθηκαν από την αντικατάσταση και όχι τον αρχικό κώδικα.

Ο υπολογισμός ομοιότητας δύο αρχείων με βάση την παραπάνω διαδικασία μπορεί να αναλυθεί στα ακόλουθα βήματα [29]:

1. Προεπεξεργασία. Αρχικά ανιχνεύονται μέθοδοι μετασχηματισμού του πηγαίου κώδικα που μπορεί να έχει εφαρμόσει ένας προγραμματιστής κατά την αντιγραφή κώδικα τρίτου. Για παράδειγμα, μπορεί να έχει πραγματοποιηθεί μεταβολή σχολίων, διαχωρισμός ή συγχώνευση δηλώσεων μεταβλητών, αλλαγή της σειράς των μεταβλητών ή/και προσθήκη ορισμένων περιττών δηλώσεων. Αυτές οι πρακτικές αποσκοπούν στην κάλυψη των λογοκλοπών. Κατά την προεπεξεργασία, όλα τα σχόλια αφαιρούνται από τον πηγαίο κώδικα και τα ονόματα των μεταβλητών και των συναρτήσεων αντικαθίστανται από απλά ονόματα. Επίσης αφαιρούνται και τα ονόματα των πακέτων στα οποία ανήκουν οι μεταβλητές και οι συναρτήσεις, καθώς και οι δηλώσεις εισαγωγής αφαιρούνται. Τέλος, οι μεταβλητές στις δηλώσεις ομαδοποιούνται ανά τύπο.

2. Μετατροπή σε σύμβολα. Στη συνέχεια ο πηγαίος κώδικας μετατρέπεται σε σύμβολα. Τα σύμβολα επιλέγονται με τέτοιο τρόπο ώστε να χαρακτηρίζουν την ουσία του προγράμματος. Αυτό συμβαίνει για να εξουδετερωθούν τροποποιήσεις που ενδεχομένως έχουν εφαρμοστεί, όπως η αλλαγή της μορφοποίησης

πηγαίου κώδικα, η μετατροπή των εξόδων, η αλλαγή ονομάτων μεταβλητών και συναρτήσεων, η μετάφραση σε διαφορετική γλώσσα και η τροποποίηση σταθερών τιμών. Με αυτόν τον τρόπο προετοιμαζεται ο κώδικας ώστε η έξοδος που θα προκύψει από αυτή τη διαδικασία να είναι κατάλληλη για σύγκριση από τον αντίστοιχο αλγόριθμο. Ένας αλγόριθμος μετατροπής κώδικα σε σύμβολα αντικαθιστά τα αναγνωριστικά με τα κατάλληλα σύμβολα με βάσει τον τύπο τους. Για παράδειγμα, όλα τα αναγνωριστικά και οι τιμές των αριθμητικών τύπων μιας γλώσσας προγραμματισμού λαμβάνουν το ίδιο σύμβολο.

3. Αποκλεισμός. Σε αυτό το σημείο, αφαιρούνται τμήματα κώδικα που είναι όμοια μεταξύ πολλών προγραμμάτων. Η μη αφαίρεση αυτών των τμημάτων ενδέχεται να οδηγήσει σε ψευδώς θετικά αποτελέσματα. Τέτοια τμήματα κώδικα αποτελούν οι setters και getters μιας κλάσης, καθώς και πρότυπα κώδικα που μπορεί να δόθηκαν στους προγραμματιστές από κάποιον εκπαιδευτή.

4. Μέτρηση ομοιότητας. Τέλος, εφαρμόζεται κάποιος αλγόριθμος υπολογισμού ομοιότητας. Ο αλγόριθμος αυτός βασίζεται συνήθως στο συνεχόμενο αριθμό συμβόλων που είναι πανομοιότυπα στα δύο αρχεία πηγαίου κώδικα. Ένας αλγόριθμος που χρησιμοποιείται από αρκετές μεθόδους [30, 31] είναι ο αλγόριθμος Greedy String Tiling [32].

Κατά τον αλγόριθμο αυτό (βλπ. Αλγόριθμος 2), η μεγαλύτερη δυνατή τιμή ομοιότητας δύο συμβολοσειρών είναι ίση με το μέγεθος μιας συμβολοσειράς. Αυτό συμβαίνει στην περίπτωση που οι υπο μελέτη συμβολοσειρές είναι πανομοιότυπες. Η μικρότερη δυνατή τιμή είναι μηδέν, στην περίπτωση που οι συμβολοσειρές δεν ταιριάζουν καθόλου.

Αλγόριθμος 2 Greedy String Tiling

[1] Αρχικά, θεωρείται ότι το σύνολο των συμβόλων που συμμετέχουν στη μέγιστη αντιστοίχιση είναι μηδέν.

[2] Στη συνέχεια, ορίζεται η μεταβλητή μέγιστης αντιστοίχισης $maxmatch$ ίση με ένα ελάχιστο μήκος.

[3] Το πρώτο σύμβολο που δεν είναι μαρκαρισμένο λαμβάνεται από τις δύο συμβολοσειρές. Εξετάζεται αν αυτά τα δύο ταυτίζονται μεταξύ τους. Η διαδικασία αυτή συνεχίζει και για τα ακόλουθα σύμβολα εώς ότου βρεθούν είτε δύο σύμβολα που δεν ταυτίζονται είτε ένα σύμβολο το οποίο είναι μαρκαρισμένο.

[4] Αν ο αριθμός των ταυτόσημων συμβόλων που προέκυψε από το προηγούμενο βήμα είναι ίσος με το $maxmatch$ του βήματος 1, τότε αποθηκεύονται οι θέσεις των πρώτων συμβόλων των δύο συμβολοσειρών από τις οποίες ξεκίνησε ο έλεγχος. Επίσης, το μήκος της συμβολοσειράς αποθηκεύεται σε μία λίστα. Σε περίπτωση που ο αριθμός των ταυτόσημων συμβόλων είναι μεγαλύτερος του $maxmatch$ τότε η τιμή της μεταβλητής ορίζεται ίση με τον αριθμό των ταυτόσημων αυτών συμβόλων. Έτσι, δημιουργείται μια νέα λίστα ώστε να αποθηκεύονται οι αντιστοχίσεις του νέου μήκους $maxmatch$.

[5] Για κάθε αντιστοίχιση που αποθηκεύτηκε στη λίστα μαρκάρονται όλα τα σύμβολα. Το μέγεθος των μαρκαρισμένων συμβόλων αυξάνεται κατά $maxmatch$ επί τον αριθμό των αντιστοιχίσεων που περιέχει η λίστα.

[6] Στη συνέχεια επαναλαμβάνεται ο αλγόριθμος από το βήμα 2 και έπειτα, εώς ότου η τιμή της μέγιστης αντιστοίχισης ισούται με την ελάχιστη τιμή μήκους αντιστοίχισης που τέθηκε αρχικά.

Για παράδειγμα, έστω ότι συμβολίζουμε με P την πρότυπη συμβολοσειρά και με T την συμβολοσειρά του κειμένου που ελέγχουμε. Η μέγιστη αντιστοίχιση παρουσίαζεται όταν μια υποσυμβολοσειρά P_p της πρότυπης συμβολοσειράς που ξεκινά από τη θέση p αντιστοιχίζεται στοιχείο προς στοιχείο με την υποσυμβολοσειρά T_t της συμβολοσειράς κειμένου που ξεκινά από τη θέση t. Η αντιστοίχιση θεωρείται ότι είναι όσο το δυνατόν μεγαλύτερη. Για την ακρίβεια, επαναλαμβάνεται μέχρι να προκύψει μια αναντιστοιχία μεταξύ δύο στοιχείων των δύο υποσυμβολοσειρών. Οι μέγιστες αντιστοιχίσεις είναι προσωρινές και όχι μοναδικά συσχετίσμένες. Δηλαδή, μια υποσυμβολοσειρά που εμπλέκεται σε μια μέγιστη αντιστοίχιση μπορεί να αποτελεί μέρος πολλών άλλων μέγιστων αντιστοιχίσεων.

Ένα πλακίδιο (tile) είναι ένας σταθερός και μοναδικός (ένα προς ένα) συνδυασμός μιας υποσυμβολειράς από το P με μια αντίστοιχη υποσυμβολοσειρά από το T . Στη διαδικασία σχηματισμού ενός πλακιδίου από μια μέγιστη αντιστοίχιση σημειώνονται (μαρκάρονται) τα σύμβολα των δύο υποσυμβολοσειρών. Αυτό συμβαίνει έτσι ώστε να μην είναι διαθέσιμα για περαιτέρω αντιστοιχίσεις.

Το ελάχιστο μήκος ορίζεται έτσι ώστε να αγνοούνται οι αντιστοιχίσεις κάτω από αυτό το μήκος. Το μήκος ελάχιστης αντιστοίχισης μπορεί να είναι ίσο με ένα. Γενικά, είναι ακέραιος μεγαλύτερος του ένα.

Σε αυτό το βήμα μπορούν να χρησιμοποιηθούν περισσότεροι του ενός αλγόριθμοι υπολογισμού ομοιότητας. Συνήθως, αυτό συμβαίνει ώστε το αποτέλεσμα της τελικής ομοιότητας του πηγαίου κώδικα να είναι αξιόπιστο.

5. Υπολογισμός τελικής ομοιότητας. Η τελευταία φάση αυτής της διαδικασίας ανίχνευσης ομοιότητας είναι ο τελικός υπολογισμός ομοιότητας. Η τελική ομοιότητα βασίζεται στις μετρικές ομοιότητας που εξήγαγαν οι αλγόριθμοι σύγκρισης που χρησιμοποιήθηκαν και στους συντελεστές βάρους τους. Το συνολικό μέτρο ομοιότητας μεταξύ δύο αρχείων πηγαίου κώδικα a και b υπολογίζεται με τον ακόλουθο τύπο:

$$sim(a,b) = \sum_{i}^{n} \frac{w_i \cdot sim_i(a,b)}{\sum_i^n w_i} \tag{2.3}$$

όπου $sim_i(a,b)$ είναι η τιμή της μετρικής ομοιότητας που λαμβάνεται από τον i-οστό αλγόριθμο ανίχνευσης ομοιότητας που υπολογίστηκε στο πρηγούμενο βήμα, w_i είναι ο συντελεστής βάρους του i-οστού αλγορίθμου ανίχνευσης ομοιότητας, και n είναι ο αριθμός όλων των χρησιμοποιούμενων αλγορίθμων. Ο συντελεστής βάρους εξαρτάται από τη σημαντικότητα του εκάστοτε αλγορίθμου για τον υπολογισμό της τελικής ομοιότητας. Για παράδειγμα, ένας αλγόριθμος που θεωρείται ότι παρέχει μια ολοκληρωμένη σύγκριση, κρίνεται ότι θα πρέπει να έχει μεγαλύτερη συμβολή στην τελική ομοιότητα και, επομένως, ο συντελεστής βάρους θα έχει μεγαλύτερη τιμή από έναν άλλο αλγόριθμο που η σύγκρισή του θεωρείται λιγότερο ολοκληρωμένη.

2.3 Υπολογισμός ομοιότητας πηγαίου κώδικα με χρήση μεθόδων επεξεργασίας κειμένου και σύγκρισης διανυσμάτων

Πολλές φορές, για τον υπολογισμό της ομοιότητας πηγαίου κώδικα χρησιμοποιούνται μέθοδοι που υπολογίζουν την ομοιότητα σε απλό κείμενο. Σε αυτή την περίπτωση ο κώδικας θεωρείται απλό κείμενο και ο αλγόριθμος υπολογισμού ομοιότητας δε λαμβάνει υπόψη τη δομή του προγράμματος [16]. Τέτοια συστήματα συνήθως αναπαριστούν το κείμενο ως σακούλα λέξεων (bag of words), από την οποία προκύπτει ένα αντιπροσωπευτικό διάνυσμα. Σε αυτή τη μέθοδο λαμβάνονται περισσότερο υπόψη η ποικιλία των λέξεων που χρησιμοποιούνται ως ονόματα συναρτήσεων, κλάσεων και μεταβλητών ή που συμμετέχουν στον σχολιασμό των οντοτήτων.

Αρχικά, το κείμενο διασπάται σε λέξεις οι οποίες σχηματίζουν μία σακούλα. Κατά την αναπαράσταση των αρχείων με αυτόν τον τρόπο, εμφανίζονται λέξεις στα αρχεία πηγαίου κώδικα με πολλές μορφολογικές παραλλαγές. Στις περισσότερες περιπτώσεις οι μορφολογικές παραλλαγές λέξεων έχουν παρόμοιες σημασιολογικές ερμηνείες και μπορούν να θεωρηθούν ισοδύναμες. Επομένως είναι απαραίτητη η χρήση αλγορίθμων που εξάγουν τις ρίζες των λέξεων αυτών. Αυτό επιτυγχάνεται με διαδικασίες θεματοποίησης (stemming), οι οποίες αφαιρούν τα προθέματα και τα επιθήματα των λέξεων. Οι θεματοποιητές (stemmers) που χρησιμοποιούνται συνήθως είναι ο Porter [33, 34] και Lovin [35].

Στη συνέχεια, όλα τα αρχεία πηγαίου κώδικα αναπαρίστανται ως διανύσματα σε ένα κοινό χώρο διανυσμάτων αφότου αφαιρεθούν πρώτα οι λέξεις χωρίς σημασιολογικό περιεχόμενο. Συγκεκριμένα, κάθε όρος του διανύσματος αντιστοιχεί σε μία ρίζα. Η απλούστερη κωδικοποίηση αρχείων πηγαίου κώδικα σε διανύσματα είναι η χρήση διανυσμάτων δυαδικών όρων. Έτσι, ένας όρος του διανύσματος έχει την τιμή ένα εάν η αντίστοιχη ρίζα υπάρχει στη σακούλα λέξεων που αντιστοιχεί στο αρχείο. Εάν η ρίζα δεν υπάρχει, έχει την τιμή μηδέν. Επειδή, η συγκεκριμένη αναπαράσταση μπορεί να οδηγήσει σε εσφαλμένα αποτελέσματα, καθώς δε λαμβάνεται υπόψη η συχνότητα εμφάνισης της λέξης χρησιμοποιούνται συστήματα σταθμίσεων [36]. Τα βάρη αντανακλούν τη σημασία μιας λέξης σε ένα συγκεκριμένο αρχείο πηγαίου κώδικα. Τα μεγάλα βάρη αποδίδονται σε έναν όρο-ρίζα που χρησιμοποιείται συχνά σε σχετικά αρχεία, αλλά σπάνια σε ολα τα υπο εξέταση αρχεία πηγαίου κώδικα και αντιπροσωπεύεται από τη συχνότητα όρου $d_{tf} = [tf_1, tf_2, ..., tf_D]$ όπου tf_i είναι η συχνότητα του όρου i στο αρχείο και D είναι ο συνολικός αριθμός μοναδικών όρων στη σακούλα λέξεων.

Όροι που εμφανίζονται σε λίγα αρχεία είναι χρήσιμοι για την διάκριση αυτών από την υπόλοιπη συλλογή. Αντιθέτως, όροι με πολύ μικρή συχνότητα που βρίσκονται σε όλα τα αρχεία θεωρούνται ότι δεν προσφέρουν στην διαδικασία εύρεσης ομοιότητας και αφαιρούνται. Η αντίστροφη στάθμιση συχνότητας αρχείων χρησιμοποιείται για την εκχώρηση υψηλότερων βαρών στις πιο διακριτικές λέξεις. Η τιμή της αντίστροφης συχνότητας αρχείων ορίζεται μέσω του κλάσματος $\frac{N}{n_i}$, όπου N είναι ο συνολικός αριθμός των αρχείων πηγαίου κώδικα στη συλλογή και n_i είναι ο αριθμός των αρχείων στα οποία εμφανίζεται ο όρος i.

Λόγω του μεγάλου αριθμού αρχείων πηγαίου κώδικα σε πολλές συλλογές, το μέτρο αυτό συνήθως περιλαμβάνει μια λογαριθμική συνάρτηση. Αυτό συμβαίνει ώστε να γίνει μια κανονικοποίηση του μέτρου

και να μην επισκιάζει τη συχνότητα όρων tf όταν συνδυάζονται αυτές οι δύο. Ο προκύπτων ορισμός της αντίστροφης συχνότητας αρχείων είναι:

$$idf_i = \log \frac{N}{n_i} \tag{2.4α'}$$

Συνδυάζοντας τη συχνότητα των όρων με την αντίστροφη στάθμιση, προκύπτει ένα σχήμα γνωστό ως tf-idf.

$$tfidf_{i,j} = tf_{i,j} \cdot idf_j \tag{2.4β'}$$

Έτσι, η αναπαράσταση συχνότητας όρου-αντίστροφης στάθμισης όρου ενός αρχείου d που αποτελείται από τους όρους $1, 2, \cdots, D$ προκύπτει ως:

$$d_{tf-idf} = [tf_1 \cdot \log \frac{n}{df_1}, tf_2 \cdot \log \frac{n}{df_2}, \cdots, tf_D \cdot \log \frac{n}{df_D}] \tag{2.4γ'}$$

Για να ληφθούν υπόψη τα αρχεία με διαφορετικά μήκη, κάθε διάνυσμα αρχείου κανονικοποιείται σε ένα μοναδιαίο διάνυσμα, του οποίου το μέτρο είναι ίσο με ένα. Χρησιμοποιώντας τις διανυσματικές αναπαραστάσεις των αρχείων μπορεί να υπολογίζεται η ομοιότητά τους με βάση κάποια μετρική, όπως η συνημιτονική ομοιότητα, η ομοιότητα Tanimoto, η ευκλείδια ομοιότητα και ο συντελεστής Pearson.

Συνημιτονική Ομοιότητα. Η συνηθέστερα χρησιμοποιούμενη είναι η ομοιότητα συνημιτόνου. Για δύο αρχεία d_i και d_j, η ομοιότητα μεταξύ τους μπορεί να υπολογιστεί ως:

$$cos(d_i, d_j) = \frac{d_i \cdot d_j}{\|d_i\| \|d_j\|} \tag{2.5}$$

Όταν η τιμή συνημιτόνου είναι ένα τα δύο αρχεία είναι πανομοιότυπα. Αν δεν υπάρχει τίποτα κοινό μεταξύ τους (δηλαδή τα διανύσματα των αρχείων είναι ορθογώνια μεταξύ τους) είναι μηδέν.

Ομοιότητα tanimoto. Η ομοιότητα Tanimoto, δε συγκρίνει μόνο τη γωνία δύο διανυσμάτων όπως συνέβαινε με τη συνημιτονική ομοιότητα. Συγκεκριμένα, δύο διανύσματα δεν αρκεί να είναι παράλληλα για να τους αποδοθεί η μέγιστη ομοιότητα αλλά να έχουν τους ίδιους όρους. Συγκεκριμένα, ο τύπος υπολογισμού της tanimoto ομοιότητας δύο διανυσμάτων είναι:

$$tsim(A, B) = \frac{\sum_i A_i \cdot B_i}{\sum_i \|A_i\|^2 + \sum_i \|B_i\|^2 - \sum_i A_i \cdot B_i} \tag{2.6}$$

Ευκλείδια Ομοιότητα. Αυτός είναι ο πιο διαισθητικός τρόπος υπολογισμού μιας απόστασης μεταξύ δύο αρχείων. Λαμβάνει υπόψη τη μετεξύ τους διαφορά απευθείας, βάσει του μεγέθους των αλλαγών στα επίπεδα του αρχείου. Αυτός ο τύπος απόστασης χρησιμοποιείται συνήθως για σύνολα δεδομένων που είναι κατάλληλα κανονικοποιημένα.

$$edis(A, B) = \sqrt{\sum_i (A_i - B_i)^2} \tag{2.7α'}$$

$$esim(A, B) = 1 - \sqrt{\sum_i (A_i - B_i)^2} \tag{2.7β'}$$

Συντελεστής Συσχέτισης Pearson. Αυτή η απόσταση βασίζεται στο συντελεστή συσχέτισης Pearson. Ο συντελεστής αυτός υπολογίζεται από τις τιμές ενός δείγματος και τις τυπικές αποκλίσεις τους. Ο συντελεστής συσχέτισης r παίρνει τιμές από -1 (μέγιστη αρνητική συσχέτιση) έως +1 (μέγιστη θετική συσχέτιση). Η απόσταση Pearson υπολογίζεται ως $dp = 1 - r$ και παίρνει τιμή 0 όταν ο συντελεστής συσχέτισης είναι +1 και 2 όταν ο συντελεστής συσχέτισης είναι -1. Ο συντελεστής συσχέτισης r υπολογίζεται ως εξής [16]:

$$r = \frac{\sum_{i=1}^{m} m \cdot A_i \cdot B_i - \sum_{i=1}^{m} A_i \cdot \sum_{i=1}^{m} B_i}{\sqrt{[m \sum_{i=1}^{m} A_i^2 - (\sum_{i=1}^{m} A_i)^2][m \sum_{i=1}^{m} B_i^2 - (\sum_{i=1}^{m} B_i)^2]}} \tag{2.8}$$

Για τον συντελεστή Pearson, που κυμαίνεται από -1 έως +1, εφαρμόζεται η κανονικοποίηση $dp = 1 - r$ αν $r \geq 0$ και $d = |r|$ αν $r < 0$.

2.4 Υπολογισμός ομοιότητας με χρήση δέντρων

Πολλά εργαλεία για τον υπολογισμό της ομοιότητας μεταξύ προγραμμάτων βασίζονται σε δέντρα που περιγράφουν τις συσχετίσεις του πηγαίου κώδικα. Τα δέντρα μπορεί να είναι δέντρα κλήσης [19], αφηρημένα συντακτικά δέντρα [20] ή δέντρα ανάλυσης [37, 38]. Η διαφορά μεταξύ αυτών είναι ότι τα δέντρα κλήσης απεικονίζουν τον τρόπο επικοινωνίας των συναρτήσεων μεταξύ τους ενώ τα υπόλοιπα αναπαριστούν τη σύνταξη μιας γλώσσας προγραμματισμού ως μια ιεραρχική δομή που μοιάζει με δέντρο.

Το βασικό πρόβλημα στην ανίχνευση όμοιων προγραμμάτων είναι η ανακάλυψη τμημάτων κώδικα που υλοποιούν την ίδια λειτουργία. Για να γίνει αυτό, πρέπει πρώτα να διασπαστεί το πρόγραμμα σε τμήματα που μπορούν να συγκριθούν. Κατόπιν θα πρέπει να καθοριστεί αν τα ζεύγη τμημάτων κώδικα είναι ισοδύναμα. Αν διασπαστεί το πρόγραμμα σε λέξεις οι οποίες μετατραπούν σε σύμβολα η διαδικασία είναι ίδια με αυτή που περιγράφηκε στο υποκεφάλαιο 3.2. Αν διασπαστεί το πρόγραμμα σε μπλοκ, το μέγεθος του τμήματος αυτού είναι πολύ μικρό ώστε το αποτέλεσμα που θα προκύψει να είναι αξιόπιστο. Επομένως, ο αλγόριθμος δε θα είναι αποδοτικός. Οπότε συνήθως, επιλέγεται να χωρίσουμε το πρόγραμμα σε συναρτήσεις [37].

Τα τμήματα κώδικα που πρέπει να συγκριθούν αρχικά μετατρέπονται σε μια μορφή δέντρου ανάλογα με τον αλγόριθμο σύγκρισης που χρησιμοποιούμε. Στην περίπτωση των αφηρημένων δέντρων και των δέντρων ανάλυσης, ένας κόμβος υποδηλώνει κάποιο στοιχείο το οποίο μπορεί να είναι είτε ένα σύμβολο, όπως ένα όνομα μιας μεταβλητής, είτε ένα μη τερματικό στοιχείο που αντιπροσωπεύει μια υποδομή, όπως μια έκφραση. Στην περίπτωση των δέντρων κλήσης κάθε κόμβος αναπαριστά μια συνάρτηση. Στη συνέχεια, εφαρμόζεται κάποιος αλγόριθμος που αναλαμβάνει τη σύγκριση των κόμβων των δύο δέντρων και κατά επέκταση την μεταξύ τους αντιστοίχιση. Ένας κόμβος δέντρου που δεν έχει αντίστοιχο κόμβο στο δεύτερο δέντρο θεωρείται ότι απουσιάζει από αυτό και ως εκ τούτου θεωρείται ότι είναι μια διαφορά μεταξύ του κώδικα. Ένας κόμβος ενός δέντρου που έχει αντίστοιχο κόμβο στο δεύτερο δέντρο αλλά περιέχει ένα διαφορετικό σύμβολο στον κόμβο αυτό θεωρείται επίσης ως μια διαφορά.

Δεδομένου ότι υπάρχουν δύο δέντρα, ο αλγόριθμος αντιστοίχισης μπορεί να βρει ένα σύνολο ζευγών κόμβων, ένα από κάθε δέντρο, έτσι ώστε κάθε κόμβος να μπορεί να εμφανιστεί σε ένα ζεύγος. Επίσης, οι κόμβοι που βρίσκονται στο ίδιο ζεύγος πρέπει να περιέχουν ταυτόσημα σύμβολα. Ακόμη, η σχέση γονέα-παιδιού καθώς η σειρά μεταξύ των αδελφών-κόμβων πρέπει να διατηρείται. Δηλαδή, πρέπει να πληρούνται οι εξής δύο προϋποθέσεις: (1) δύο κόμβοι μπορούν να ταιριάζουν μόνο αν οι γονείς τους ταιριάζουν. (2) ας υποθέσουμε ότι ο v_1 αντιστοιχεί στο v_2, ο w_1 αντιστοιχεί στο w_2, v_1 και w_2 είναι αδέλφια, και v_2 και w_2 είναι επίσης αδέλφια. Στη συνέχεια, θα εμφανιστεί το v_1 πριν από το w_1 εάν και μόνο εάν το v_2 εμφανίζεται πριν από το w_2.

Η ομοιότητα δύο δέντρων ισούται με το άθροισμα των βαρών κάθε ζεύγους αντιστοιχισμένων κόμβων. Το βάρος ενός ζεύγους κόμβων είναι ένα, εάν περιέχουν ταυτόσημα σύμβολα ή μηδέν διαφορετικά.

Η σύγκριση δύο προγραμμάτων ανάγεται σε σύγκριση πολλών τμημάτων κώδικα. Αυτό με τη σειρά του ισοδυναμεί σε σύγκριση πολλών δέντρων μεταξύ τους. Άρα η πολυπλοκότητα είναι αυξημένη. Για παράδειγμα, αν το πρόγραμμα A δημιουργεί n υποδέντρα , ο αριθμός των κόμβων είναι ικανοποιητικά μεγάλος (συγκεκριμένα μεγαλύτερος από ένα κατώτατο όριο κόμβων που πρέπει να έχει ένα δέντρο για να είναι συγκρίσιμο), και ο B παράγει m υποδέντρα που ο αριθμός των κόμβων είναι εξίσου ικανοποιητικά μεγάλος, τότε απαιτούνται mn συγκρίσεις. Για να μειωθεί ο χρόνος αυτής της διαδικασίας σύγκρισης χτίζεται ένας πίνακας κατακερματισμού για κάθε υποδέντρο του προγράμματος A. Μόνο τα υποδέντρα των οποίων οι τιμές κατακερματισμού είναι ίδιες με τις τιμές κατακερματισμού του a_1 μπορούν να συγκριθούν με το a_1. Για τα προγράμματα αυτά, εάν $m \geq n$, ο αριθμός των όμοιων υποδέντρων που ανιχνεύονται μπορεί να είναι μέχρι n. Έτσι, η πολυπλοκότητα χρόνου μειώνεται από $O(n^3)$ σε $O(n^2)$. Το σκορ ομοιότητας δύο προγραμμάτων αν $S_1, S_2, \cdots, S_n$, είναι οι τιμές ομοιότητας των n υποδέντρων υπολογίζεται από τη σχέση:

$$treeSum = \frac{S_1 + S_2 + ... + S_n}{n} \tag{2.9}$$

3 Σχετική Βιβλιογραφία

Στο παρόν κεφάλαιο αναλύονται μέθοδοι υπολογισμού ομοιότητας δύο ή περισσότερων αρχείων πηγαίου κώδικα οι οποίες βρίσκονται στη βιβλιογραφία.

Τα συστήματα υπολογισμού ομοιότητας πηγαίου κώδικα της βιβλιογραφίας χρησιμοποιούνται κυρίως για την αντιμετώπιση προγραμματιστικής λογοκλοπής και την αναγνώση επαναχρησιμοποιημένου κώδικα. Όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο, ο υπολογισμός της ομοιότητας μπορεί να πραγματοποιηθεί είτε αναλύοντας των κώδικα με εργαλεία που τον αντιμετωπίζουν ως κείμενο είτε λαμβάνοντας υπόψη και τη δομή του προγράμματος. Παρόλες τις επιμέρους διαφορές τους, οι σχετικές μέθοδοι ακολουθούν μια κοινή λογική στον τρόπο με τον οποίο αντιμετωπίζουν το πρόβλημα.

Η λογική αυτή ξεκινά με την εισαγωγή αρχείων πηγαίου κώδικα στο σύστημα και την ανάλυσή τους ώστε να εξάγουν χαρακτηριστικά τα οποία είναι χρήσιμα για σύγκριση. Στη συνέχεια προτείνονται αλγόριθμοι οι οποίοι εξάγουν τιμές ομοιότητας με βάση τα χαρακτηριστικά που τους δίνονται ως είσοδοι. Έτσι, οι αλγόριθμοι αυτοί αποφασίζουν για το αν δύο αρχεία είναι όμοια μεταξύ τους. Τα βήματα αυτά αναλύονται λεπτομερώς στη συνέχεια αυτού του κεφαλαίου.

3.1 Εισαγωγή αρχείων πηγαίου κώδικα και εξαγωγή χαρακτηριστικών

Η πρώτη φάση περιλαμβάνει την εισαγωγή των υπό εξέταση αρχείων πηγαίου κώδικα στο σύστημα. Τα περισσότερα συστήματα δεν εφαρμόζουν κάποιο αλγόριθμο στον αρχικό πηγαίο κώδικα. Απαιτείται πρώτα μια προεργασία ουτώς ώστε να προκύψουν κάποια χαρακτηριστικά τα οποία θα είναι αξιοποιήσιμα από το σύστημα.

Αρχικά, τα συστήματα που αντιμετωπίζουν τα αρχεία πηγαίου κώδικα ως κείμενο [16, 18] και εκείνα που βασίζονται στη χρήση συμβόλων έχουν κοινή διαδικασία προεπεξεργασίας [39]. Αυτή περιλαμβάνει:

- Αφαίρεση σχολίων και συμβολοσειρών προς εκτύπωση
- Μετατροπή των κεφαλαίων γραμμάτων σε μικρά
- Αφαίρεση γραμμάτων που δεν ανήκουν στα αποδεκτά αναγνωριστικά

Τα σχόλια και οι συμβολοσειρές προς εκτύπωση θεωρείται ότι δε συνεισφέρουν στον υπολογισμό ομοιότητας δύο προγραμμάτων. Αντιθέτως, επικρατεί η πεποίθηση ότι αλλοιώνουν την τελική ομοιότητα των προγραμμάτων και επομένως αφαιρούνται.

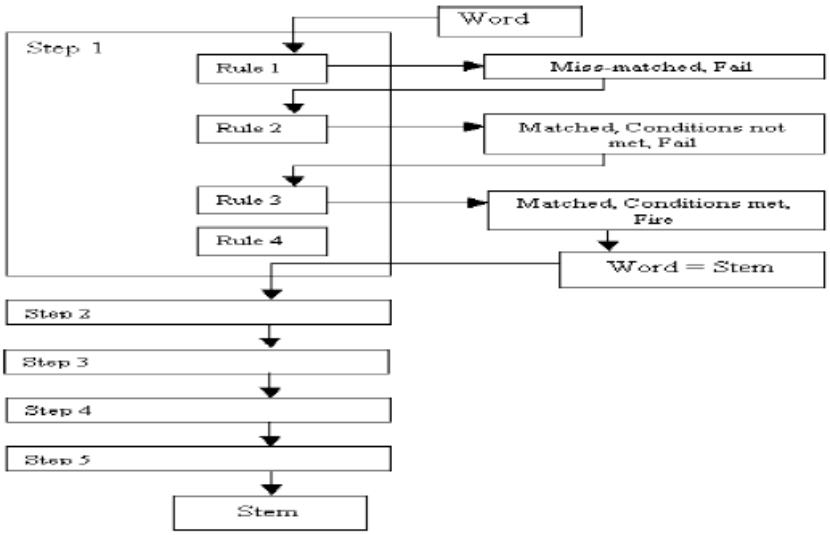
Κατά τη μέθοδο της λανθάνουσας σημασιολογικής ανάλυσης και κατά τη μέθοδο ανάλυσης του πηγαίου κώδικα ως απλό κείμενο δε μελετάται η δομή του προγράμματος και η σύνταξη αυτού. Επομένως, η συγκεκριμένη διάρθρωση των όρων που χρησιμοποιούνται παραμένει αναξιοποίητη και έτσι είναι απαραίτητος ο διαχωρισμός των αρχείων πηγαίου κώδικα σε λέξεις. Κάποιες λέξεις ενδέχεται να μην έχουν σημασιολογική διαφορά αλλά να μην έχουν και την ίδια μορφή. Για παράδειγμα, οι λέξεις imported και import θα μπορούσαν να περιγράψουν την ίδια έκφραση χωρίς ωστόσο να είναι ταυτόσημες. Επομένως, είναι απαραίτητη η χρήση αλγορίθμων ώστε να μετατρέψουν μια λέξη στη μορφή της ρίζας της. Αυτό πραγματοποιείται με τη διαδικασία θεματοποίησης μιας λέξης (stemming). Δύο αλγόριθμοι που χρησιμοποιούνται είναι ο θεματοποιητής Lovin [35] και ο θεματοποιητής Porter [33, 34].

Ο θεματοποιητής Porter αποτελείται από πέντε βήματα ώστε να προκύψει ως αποτέλεσμα η ρίζα μιας λέξης. Σε κάθε βήμα πραγματοποιείται ένας έλεγχος κατά τον οποίο διαπιστώνεται εάν μια συγκεκριμένη κατάληξη ταιριάζει με μια λέξη. Στην περίπτωση που ο έλεγχος είναι θετικός, η κατάληξη απομακρύνεται. Έτσι, η λέξη λαμβάνει τη μορφή της ρίζας σύμφωνα με τους κανόνες που επιτάσσει ο κάθε έλεγχος που πραγματοποιεί ο αλγόριθμος αυτός. Παρακάτω στο διάγραμμα απεικονίζεται η διαδικασία κατά την οποία λαμβάνεται η ρίζα μιας λέξης μέσω του αλγορίθμου αυτού.

Αν συμβολίσουμε με C φωνήεντα και V τα σύμφωνα, οποιαδήποτε λέξη της αγγλικής μπορεί να αναπρασταθεί [16] με τον ενιαίο τύπο:

$$[C](VC)^m[V] \tag{3.1}$$

όπου m υποδηλώνει οποιονδήποτε αριθμό επαναλήψεων της παράστασης VC και οι αγκύλες την προαιρετική παρουσία του περιεχομένου τους. Η τιμή m ονομάζεται μέτρο μιας λέξης και μπορεί να πάρει οποιαδήποτε τιμή μεγαλύτερη ή ίση με το μηδέν. Χρησιμοποιείται για να αποφασίσει αν πρέπει να αφαιρεθεί ένα συγκεκριμένο επίθημα. Όλοι αυτοί οι κανόνες είναι της μορφής: (συνθήκη) $S1 \rightarrow S2$. Το επίθημα που βρίσκεται αριστερά του βέλους αντικαθίσταται από το επίθημα στα δεξιά του, εάν τα υπόλοιπα γράμματα του $S1$ ικανοποιούν τη συνθήκη.



Σχήμα 1: Θεματοποίηση λέξης

Το πρώτο βήμα του αλγορίθμου έχει σχεδιαστεί για να ασχολείται με τις παρελθοντικές μετοχές και τον πληθυντικό αριθμό. Αυτό το βήμα είναι το πιο πολύπλοκο και χωρίζεται σε τρία μέρη στον αρχικό ορισμό, 1a, 1b και 1c. Το πρώτο μέρος ασχολείται με τον πληθυντικό αριθμό, για παράδειγμα πραγματοποιούνται οι αλλαγές όπως $sses \rightarrow ss$ ή αφαιρείται η κατάληξη s. Το δεύτερο μέρος αφαιρεί τα -ed και τα -ing ή μετατρέπει το -eed σε ee, όπου χρειάζεται. Το δεύτερο μέρος συνεχίζεται μόνο εάν απομακρυνθεί το -ed ή το -ing και μετασχηματιστεί το υπόλοιπο τμήμα για να βεβαιωθεί ότι ορισμένα επιθήματα θα αναγνωριστούν αργότερα. Το τρίτο μέρος μετατρέπει ένα τερματικό y σε ένα i.

Τα υπόλοιπα βήματα είναι σχετικά απλά και περιέχουν κανόνες για την αντιμετώπιση διαφορετικών τάξεων επιθημάτων. Αρχικά μετασχηματίζει τα διπλά επιθήματα σε ένα επίθημα. Στη συνέχεια γίνεται η αφαίρεση και του τελικού επιθήματος, εφόσον πληρούνται οι σχετικές προϋποθέσεις.

Στην περίπτωση που το σύστημα λαμβάνει υπόψη και τη δομή του πηγαίου κώδικα, όπως συμβαίνει με τα συστήματα χρήσης συμβόλων, το στάδιο της προεπεξεργασίας δεν περιορίζεται μόνο στη λεξικολογική ανάλυση. Στα συστήματα χρήσης συμβόλων προσδιορίζονται αρχικά τα μπλοκ όλων των συναρτήσεων και των διαδικασιών που εμφανίζονται στον πηγαίο κώδικα. Εν συνεχεία, απαιτείται η αντικατάσταση των λέξεων και των εκφράσεων με μοναδικά σύμβολα. Τα σύμβολα θα πρέπει να επιλέγονται έτσι ώστε να χαρακτηρίζουν την ουσία του πηγαίου κώδικα και όχι επιφανειακές πτυχές αυτού. Για παράδειγμα, οι κενοί χαρακτήρες δε θα πρέπει να παράγουν κάποιο σύμβολο. Αυτό επιτάσσει την ανάγκη να σχηματιστεί μια λίστα με πρωτότυπα σύμβολα. Η διεργασία της τοποθέτησης συμβόλων αντί του πηγαίου κώδικα ανατίθεται συνήθως σε έναν αναλυτή. Ο αναλυτής επιτρέπει την παρουσία περισσότερων σημασιολογικών πληροφοριών στα σύμβολα που επιλέγονται για κάθε έκφραση.

Αν ένα σύστημα για τον υπολογισμό της ομοιότητας πηγαίου κώδικα στηρίζεται αποκλειστικά και μόνο στη δομή του, τότε γίνεται χρήση δέντρων. Έτσι, η διαδικασία της προεπεξεργασίας διαφέρει σε σχέση με αυτή που ακολουθείται στα συστήματα που αντιμετωπίζουν τον πηγαίο κώδικα ως απλό κείμενο ή στα συστήματα όπου γίνεται αντικατάσταση του κώδικα με σύμβολα. Αυτό συμβαίνει διότι τα υπό εξέταση αρχεία πηγαίου κώδικα θα πρέπει να μετατραπούν σε δέντρα. Ανάλογα με το είδος του δέντρου που πρέπει να παραχθεί γίνεται και η κατάλληλη επεξεργασία του πηγαίου κώδικα.

Ένα αρχείο πηγαίου κώδικα αναπαρίσταται είτε ως δέντρο ανάλυσης είτε ως αφηρημένο συντακτικό δέντρο. Η αναπαράσταση δέντρου πρέπει να αντικατοπτρίζει τη συντακτική δομή και την ιεραρχική δομή του πηγαίου κώδικα. Ένας κόμβος στο δέντρο ισοδυναμεί με ένα σύμβολο ή ένα μη τερματικό στοιχείο που αντιπροσωπεύει μια υποδομή. Η συντακτική δομή χρησιμοποιείται για να αποφευχθεί η αντιστοίχιση δύο μη συμβατών δομών. Τέτοιες μη συμβατές δομές είναι μια δήλωση δεδομένων και μια δήλωση συνάρτησης. Η ιεραρχική δομή χρησιμοποιείται για να επιβάλει ότι δύο δομές μπορούν να ταιριάζουν μόνο αν βρίσκονται στο ίδιο επίπεδο εμφώλευσης. Παρακάτω παρουσιάζεται ο αλγόριθμος 3 που χρησιμοποιείται για την κατασκευή των δέντρων.

Στην περίπτωση που ο αλγόριθμος χρησιμοποιεί δέντρα κλήσης συναρτήσεων, η διαδικασία εξαγωγής του δέντρου που ακολουθείται διαφέρει. Για να προκύψουν τα δέντρα κλήσης συναρτήσεων πρέπει πρώτα

Αλγόριθμος 3 Αλγόριθμος κατασκευής δέντρων

[1.] Για έναν αριστερά-αναδρομικό κανόνα παραγωγής (ή ομοίως δεξιο-αναδρομικοί κανόνες) στη γραμματική, επιθυμούμε να χτίσουμε ένα πεπλατυσμένο δέντρο αντί για ένα λοξό δέντρο.
[2.] Ένα πρόβλημα με τα δέντρα αυτά είναι ότι το μέγεθος τους μπορεί να είναι υπερβολικά μεγάλο. Τα προβλήματα με μια μεγάλη αναπαράσταση δέντρου είναι ότι χρειάζεται περισσότερος χώρος μνήμης. Επίσης, περισσότερος χρόνος θα διατεθεί για την αντιστοίχιση. Επομένως, η δεύτερη κατευθυντήρια γραμμή για το σχεδιασμό της εσωτερικής αναπαράστασης δέντρων είναι η εξάλειψη όσο το δυνατόν περισσότερων ασήμαντων μη τερματικών.
[3.] Η σύγκριση μεταξύ δύο προγραμμάτων πρέπει να σέβεται την ιεραρχική δομή των προγραμμάτων.
[4.] Μερικές φορές η ίδια συντακτική δομή δηλώνεται με περισσότερα από ένα μη τερματικά σύμβολα. Αυτό συμβαίνει προκειμένου να γίνει ευκολότερη η κατασκευή του δέντρου.
[5.] Η εσωτερική αναπαράσταση δέντρων περιέχει κόμβους που αντιπροσωπεύουν σχόλια εάν αυτά λαμβάνονται υπόψη και εντολές προεπεξεργαστή.

να εξαχθεί ο γράφος κλήσης συναρτήσεων. Η διαδικασία που ακολουθείται για την εξαγωγή του γράφου αυτού παρουσιάζεται στο πλαίσιο του Αλγορίθμου 4.

Αλγόριθμος 4 Εξαγωγή γράφου κλήσης συνάρτησης

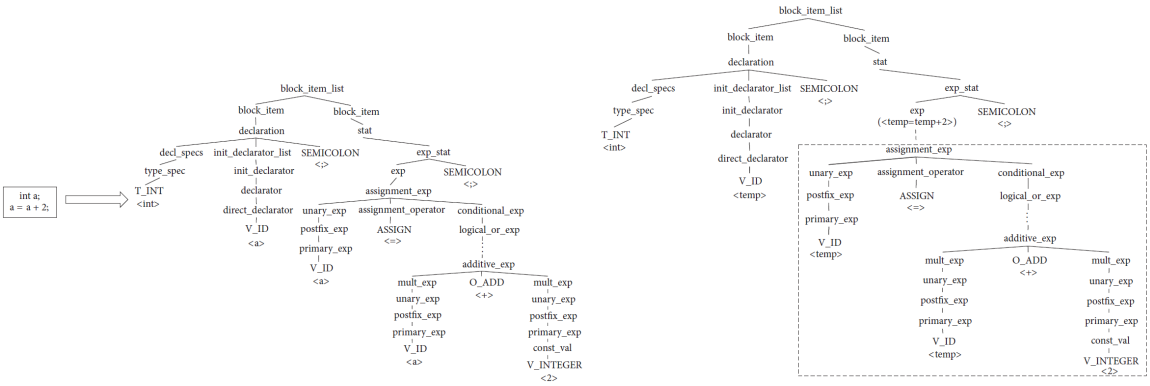
[1] Αρχικά,χρησιμοποιείται κάποιο εργαλείο για να εξαχθούν οι συναρτήσεις από τις επικαλούμενες σχέσεις. Στη συνέχεια γίνεται η χαρτογράφηση σε μια τυποποιημένη μορφή του $Fa \rightarrow Fb$. Όλες οι σχέσεις αποθηκεύονται σε ένα κείμενο, το οποίο αντιστοιχεί ένα προς ένα με τα αρχεία πηγαίου κώδικα.
[2] Στη συνέχεια, γίνεται επιλογή ενός τυχαίου αρχείου από το αποτέλεσμα του βήματος 1 και χωρίζονται οι σχέσεις κλήσης συνάρτησης στο σύνολο καλούντων και στο σύνολο καλούμενων. Για παράδειγμα, η σχέση $F_a \rightarrow F_b$ ως μορφή συνόλων θα γραφτεί ως εξής: $Set_{D1} = \{F_a\}$ και $Set_{D2} = \{F_b\}$. Επίσης, κάθε όνομα μιας συνάρτησης αντιστοιχίζεται σε έναν μοναδικό φυσικό αριθμό και έτσι αντικαθίστανται τα ονόματα. Στη συνέχεια, αποθηκεύονται τα ονόματα συναρτήσεων με τους αντίστοιχους μοναδικούς αριθμούς σε ένα ενιαίο σύνολο, για παράδειγμα $F_m = \{F_a = 1, F_b = 2\}$. Αν το όνομα της συνάρτησης υπάρχει στο F_m, λαμβάνεται η τιμή από το F_m, αλλιώς δημιουργείται μια νέα τιμή στο F_m.
[3] Τέλος, συγχονεύονται όλοι οι γράφοι κλήσης συνάρτησης που έχουν προκύψει σε έναν. Απαριθμούνται όλοι οι γράφοι από το βήμα 1, συγχωνεύονται οι αρχικές σχέσεις κλήσεων και αντικαθίστανται τα ονόματα των συναρτήσεων με τους μοναδικούς αριθμούς όπως ορίστηκαν στο F_m.

Αν ένα σύστημα στηρίζεται σε αφηρημένα συντακτικά δέντρα τότε για την εξαγωγή των δέντρων αυτών χρησιμοποιείται ένας συντακτικός αναλυτής. Η ανάλυση σε αφηρημένα δένδρα σύνταξης εξυπηρετεί την περαιτέρω κατανόηση της σημασιολογίας. Στη συνέχεια, εφαρμόζονται κάποιες προσαρμογές σε κάθε αφηρημένο δέντρο σύνταξης. Αυτό συμβαίνει για να απλοποιηθεί η δομή του δέντρου. Επίσης, αντικαθίστανται τα ονόματα των μεταβλητών και των δηλώσεων μεγέθους των συστοιχιών με ενιαία σύμβολα, όπως περιγράφηκε και προηγουμένως. Για παράδειγμα, όλες οι μεταβλητές αντικαθίστανται με ένα κοινό σύμβολο ή οι δηλώσεις μεγέθους των πινάκων και οι δείκτες στοιχείων αντικαθίστανται με ένα επίσης κοινό σύμβολο. Έτσι, μπορούν να ανιχνευθούν τυχόν ομοιότητες σε επίπεδο μεταβλητών και να μην αγνοηθούν σε περίπτωση που χρησιμοποιείται διαφορετικό όνομα για να περιγράψει την ίδια έννοια.

Επειδή η ανατύπωση των εκφράσεων σε διαφορετικές εκφράσεις δεν είναι ασήμαντη, σπάνια υπάρχει πρόβλημα για λογοκλοπή με τροποποίηση εκφράσεων. Οι πληροφορίες της δομής των εκφράσεων που σχετίζονται με υποδέντρα στο αφηρημένο δέντρο σύνταξης δε λαμβάνονται υπόψη. Έτσι, ένα απλοποιημένο δέντρο αφηρημένης σύνταξης δεν έχει αυξημένες απαιτήσεις χρόνου όταν εκτελείται από τον αλγόριθμο. Στην εικόνα 2 παρουσίαζεται μια εντολή, το αρχικό αφηρημένο συντακτικό δέντρο με το οποίο ισοδυναμεί αυτή καθώς και η τελική μορφή του δέντρου αυτού.

Η διαδικασία που περιγράφηκε εώς τώρα εξαρτάται και από τη γλώσσα προγραμματισμού των υπό εξέταση αρχείων πηγαίου κώδικα. Κάθε εργαλείο, δηλαδή, που αναλαμβάνει την εκτέλεση είναι προσαρμοσμένο για μια συγκεκριμένη γλώσσα.

Εφόσον ολοκληρώνεται η διαδικασία της προεπεξεργασίας των αρχείων πηγαίου κώδικα που δόθηκαν ως είσοδοι σε ένα σύστημα, εξάγονται κάποια χαρακτηριστικά για κάποιες μεθόδους [16, 18]. Στην περίπτωση των μεθόδων που χρησιμοποιούν λανθάνουσα σημασιολογική ανάλυση ή στις μεθόδους επεξεργασίας κειμένου που μετατρέπουν τον κώδικα σε σακούλες λέξεων για να πραγματοποιηθεί η σύγκριση



Σχήμα 2: Παραδείγματα συντακτικών δέντρων

απαιτείται η χρήση διανυσμάτων. Ένα εργαλείο παράγει για κάθε αρχείο πηγαίου κώδικα ένα διάνυσμα. Κάθε όρος του διανύσματος αντιστοιχεί σε μια μοναδική λέξη που εμφανίζεται στις σακούλες λέξεων που αντιστοιχούν στα αρχεία αυτά. Η τιμή που περιέχει ένα διάνυσμα στη θέση i εκφράζει τον αριθμό των φορών που εμφανίζεται στο αρχείο πηγαίου κώδικα η λέξη που αντιστοιχεί στη θέση αυτή. Το εργαλείο αυτό, δίνει βάρη στάθμισης στα διανύσματα όπως αναφέρεται στο Κεφάλαιο 2 ώστε να αυξηθεί η ακρίβεια του αποτελέσματος.

Βάρη μπορούν να αποδοθούν ακόμα και σε αφηρημένα συντακτικά δέντρα αν [20] οι κόμβοι τους περιέχουν πληροφορίες που αφορούν τα αρχεία του πηγαίου κώδικα. Σε αυτή την περίπτωση, τα βάρη των κόμβων εξαρτώνται και πάλι από τη συχνότητα όρων και την αντίστροφη συχνότητα αρχείων πηγαίου κώδικα που μελετήθηκε στο Κεφάλαιο 2. Θεωρείται ότι τα βάρη στους κόμβους των αφηρημένων συντακτικών δέντρων αντικατοπτρίζουν τη σημασία των αντίστοιχων τμημάτων του πηγαίου κώδικα. Χαμηλότερα βάρη δίνονται, συνήθως, στους κόμβους που αντιπροσωπεύουν κοινές εκφράσεις που εμφανίζονται σε πολλά άλλα προγράμματα. Διαφορετικοί τύποι συμβόλων και εκφράσεων εμφανίζονται συχνά σε πολλά προγράμματα. Αυτοί δεν έχουν μεγάλη σημασιολογική έννοια και μπορούν να αντιμετωπίζονται ως λέξεις μηδενικής σημασίας (stop words).

Για κάθε υποδέντρο s ενός δέντρου T υπάρχει ένα βάρος $w(S,T)$. Η συμβολοσειρά που προκύπτει κατά την ενδοδιατεταγμένη διάσχιση στο υποδέντρο συμβολίζεται ως $word_s$. Αν η συμβολοσειρά αυτή αντιμετωπίζεται ως λέξη μηδενικής σημασίας, το βάρος του υποδέντρου αυτού προσαρμόζεται στο μηδέν. Αντίθετα, αν η προκύπτουσα συμβολοσειρά αποτελεί μια έκφραση που δεν εμφανίζεται σε όλα τα αρχεία πηγαίου κώδικα, το βάρος μπορεί να υπολογιστεί ως εξής:

$$w_{s,T} = TF(s,T) \cdot IDF(s,T) \tag{3.2}$$

όπου $TF(s,T) = \frac{cnt(s,T)}{n(T)}$, $IDF(s,T) = \log_2(1 + \frac{N}{c(s)})$ και $cnt(s,T)$ είναι η συχνότητα εμφάνισης του υποδέντρου στο αρχικό δέντρο, $n(T)$ είναι ο αριθμός των υποδέντρων στο συγκεκριμένο δέντρο και $c(s)$ είναι ο αριθμός των αφηρημένων συντακτικών δέντρων που περιέχουν το υποδέντρο. Η μεταβλητή N χρησιμοποιείται για να αναπαραστήσει τον αριθμό των αφηρημένων συντακτικών δέντρων που προεκυψαν από το πρόγραμμα και σχετίζονται με συγκεκριμένη εντολή.

3.2 Αλγόριθμοι υπολογισμού ομοιότητας

Το δεύτερο κοινό βήμα σε όλα τα συστήματα υπολογισμού ομοιότητας είναι η εφαρμογή ενός αλγορίθμου σύγκρισης. Εφόσον έχει προηγηθεί η κατάλληλη προεπεξεργασία, ο πηγαίος κώδικας έχει λάβει τη μορφή που είναι ικανή να χρησιμοποιηθεί από κάποιο αλγόριθμο σύγκρισης. Η διαδικασία που ακολουθείται σε αυτό το βήμα είναι ανεξάρτητη από τη γλώσσα προγραμματισμού που χρησιμοποιήθηκε για τη συγγραφή των υπό εξέταση αρχείων πηγαίου κώδικα.

Όσον αφορά τα συστήματα που χρησιμοποιούν τεχνικές επεξεργασίας κειμένου, εφόσον έχουν προβάλλει τα αρχεία πηγαίου κώδικα στο διανυσματικό χώρο με τη μορφή διανυσμάτων, η σύγκριση των αρχείων αυτών ανάγεται στη σύγκριση των αντίστοιχων διανυσμάτων. Επομένως, ένα εργαλείο υπολογισμού δέχεται ως είσοδο τα διανύσματα των υπό σύγκριση αρχείων και εξάγεται μια τιμή ομοιότητας. Η πιο συνηθισμένη διαδικασία σύγκρισης διανυσμάτων είναι η συνημιτονική ομοιότητα που αναφέρθηκε στο Κεφάλαιο 2.

Στην περίπτωση των συστημάτων που χρησιμοποιούν σύμβολα οι αλγόριθμοι σύγκρισης στηρίζονται κυρίως στις μέγιστες αντιστοιχίσεις μεταξύ των συμβολοσειρών που προκύπτουν. Κάθε σύμβολο της μια συμβολοσειράς συγκρίνεται με κάθε σύμβολο της άλλης χρησιμοποιώντας ένα εξωτερικό εργαλείο. Το μέτρο ομοιότητας πρέπει να αντικατοπτρίζει το κλάσμα των συμβόλων από τα πρωτότυπα αρχεία πηγαίου κώδικα που καλύπτονται από αντιστοιχίσεις. Υπάρχουν δύο λογικές επιλογές: Εάν θέλουμε μια ομοιότητα 100% να σημαίνει ότι οι δύο υπό σύγκριση συμβολοσειρές είναι ισοδύναμες, πρέπει να λάβουμε υπόψη και τις δύο στον υπολογισμό της ομοιότητας. Εάν, από την άλλη πλευρά, κάθε αρχείο πηγαίου κώδικα που αποτελεί απόσπασμα από ένα άλλο εκτενέστερο αρχείο θέλουμε να έχει ομοιότητα 100%, θα πρέπει να εξεταστεί μόνο η συντομότερη συμβολοσειρά. Για την πρώτη περίπτωση, που είναι πιο συνήθης, το ακόλουθο μέτρο ομοιότητας δύο αρχείων πηγαίου κώδικα είναι:

$$sim(A,B) = \frac{2\sum_{match(a,b,length)\in tiles} length}{\|A\| + \|B\|} \tag{3.3}$$

όπου $length$ είναι το μηκος των αντιστοιχισμένων υποσυμβολοσειρών a και b και δημιουργούν ένα πλακίδιο όπως ορίστηκε στο Κεφάλαιο 2.

Ένα άλλο σύστημα [39] για τον υπολογισμό της ομοιότητας χρησιμοποιεί τον ακόλουθο τύπο:

$$match = \frac{same\_symbols - diff\_symbols}{\min_{f\in files}|f|} - \frac{\max_{f\in files}|f| - \min_{f\in files}|f|}{\max_{f\in files}|f|} \tag{3.4}$$

όπου τα $\max_{f\in files}|f|$ και $\min_{f\in files}|f|$ είναι αντίστοιχα το μέγεθος της μεγαλύτερης και της μικρότερης συμβολοσειράς, η μεταβλητή $same\_symbols$ δηλώνει τον αριθμό κοινών συμβόλων και η μεταβλητή $diff\_symbols$ δηλώνει τον αριθμό των διαφορών μιας γραμμής μεταξύ των μπλοκ των συμβόλων αντιστοίχισης.

Ένας άλλος αλγόριθμος που χρησιμοποιείται για την εύρεση κοινών συμβόλων σε τέτοιου είδους συστήματα είναι ο αλγόριθμος levenshtein (βλπ. Αλγόριθμος 5). Ο αλγόριθμος αυτός επιστρέφει τον αριθμό των διαφορετικών συμβόλων που υπάρχουν στις δύο συμβολοσειρές. Έτσι, αν δύο αρχεία πηγαίου κώδικα είναι πανομοιότυπα, γεγονός που σημαίνει ότι και οι συμβολοσειρές που σχηματίστηκαν είναι πανομοιότυπες, η τιμή που επιστρέφεται είναι ίση με το μηδέν.

Αλγόριθμος 5 Απόσταση Levenshtein

[1.] Αρχικά δίνονται ως είσοδοι οι δύο συμβολοσειρές

[2.] Στη συνέχεια, δημιουργείται ένας πίνακας. Ο αριθμός των γραμμών ισούται με το μέγεθος της μιας συμβολοσειράς αυξημένο κατά ένα. Ο αριθμός των στηλών ισούται με το μέγεθος της άλλης συμβολοσειράς αυξημένο επίσης κατά ένα.

[3.] Στη συνέχεια, η πρώτη στήλη αρχικοποιείται με το δείκτη της κάθε σειράς, ενώ η πρώτη σειρά με το δείκτη της κάθε στήλης. Αν i είναι ο δείκτης των γραμμών και j ο δείκτης των στηλών τότε οι δείκτες αυτή αντιστοιχούν σε ένα γράμμα της αντίστοιχης συμβολοσειράς αντίστοιχα. Το $i = 1$ αντιστοιχεί στο πρώτο γράμμα της αντίστοιχης συμβολοσειράς.

[4.] Έπειτα, κάθε σύμβολο της μιας συμβολοσειράς συγκρίνεται με το κάθε σύμβολο της άλλης. Αν δύο σύμβολα είναι ταυτόσημα, μια μεταβλητή c λαμβάνει την τιμή 0. Σε κάθε άλλη περίπτωση λαμβάνει την τιμή 1. Έτσι, αν συγκρίνεται το i σύμβολο της μιας συμβολοσειράς με το j σύμβολο της άλλης, η τιμή του πίνακα στην αντίστοιχη θέση ισούται με $\min\{M[i-1,j]+1, M[i,j-1]+1, M[i,j]+c\}$

[5.]Τέλος, επιστρέφεται η τιμη που βρίσκεται $M[m,n]$ του πίνακα όπου m το μήκος της μιας συμβολοσειράς και n το μήκος της άλλης.

Ο αλγόριθμος που αναφέρθηκε παραπάνω μπορεί να προσαρμοστεί και στην περίπτωση που το υπό σύγκριση χαρακτηριστικό είναι δέντρο [37]. Ο αλγόριθμος που παρουσιάζεται παρακάτω(βλπ. Αλγόριθμος 6) υπολογίζει το βάρος μιας μέγιστης αντιστοίχισης μεταξύ δύο δέντρων. Μια αντιστοίχιση είναι ένα σύνολο ζευγών αντίστοιχων κόμβων, ένας από κάθε δένδρο, οι οποίοι πρέπει να πληρούν κάποιες προϋποθέσεις. Αρχικά, οι αντίστοιχοι κόμβοι πρέπει να περιέχουν συγκρίσιμα ή πανομοιότυπα σύμβολα, διαφορετικά είναι εντελώς ανόμοιοι μεταξύ τους και δεν έχει αξία η μεταξύ τους αντιστοίχιση. Επίσης, για κάθε ζευγάρι κόμβων που αντιστοιχίζεται μεταξύ τους, η σχέση γονέα-παιδιού καθώς και η σειρά μεταξύ των αδελφών κόμβων πρέπει να διατηρούνται. Σε αντίθετη περίπτωση, μια τέτοια διαφορά σημαίνει ότι δεν καλούνται οι συναρτήσεις διαδοχικά με τον ίδιο τρόπο αν τα υπο σύγκριση δέντρα είναι δέντρα κλήσης ή

οι δομές που αναπαριστούν τα δύο υποδέντρα απο κει και έπειτα είναι διαφορετικές αν τα υπό σύγκριση δέντρα είναι συντακτικά.

Η διαδικασία που ακολουθείται (βλπ. Αλγόριθμος 6) είναι αναδρομική. Η τιμή που επιστρέφεται εξακολουθεί να μην είναι κανονικοποιημένη. Η διαφορά όμως έγκειται στο ότι μεγάλη τιμή επιστροφής συνεπάγεται και μεγάλη ομοιότητα μεταξύ των δύο δέντρων και συνεπώς μεγαλύτερη ομοιότητα μεταξύ των αρχείων πηγαίου κώδικα που τέθηκαν υπό σύγκριση.

Αλγόριθμος 6 Σύγκριση δέντρων

[1.]Αρχικά ελέγχονται αν οι ρίζες των δύο δέντρων είναι ταυτόσημες
[2.]Στη συνέχεια, δημιουργείται ένας πίνακας. Ο αριθμός των γραμμών ισούται με τον αριθμό των κόμβων-παιδιών της ρίζας του ενός δέντρου αυξημένο κατά ένα. Ο αριθμός των στηλών ισούται με τον αριθμό των κόμβων-παιδιών της ρίζας του άλλου δέντρου αυξημένο κατά ένα.
[3.] Στη συνέχεια, αρχικοποιείται η πρώτη στήλη και η πρώτη σειρά του πίνακα με μηδέν.
[4.] Έπειτα κάθε κόμβος-παιδί i της μιας ρίζας συγκρίνεται με κάθε κόμβο-παιδί j της άλλης ρίζας.
[5.]Έτσι, η τιμή $M[i,j]$ του πίνακα ισούται με $max(M[i,j-1], M[i-1,j], M[i-1,j-1] + weight)$ όπου η μεταβλητή $weight$ είναι η τιμή που επιστρέφει ο αλγόριθμος αυτός αν δοθούν ως ορίσματα τα υποδέντρα με ρίζες τους κόμβους-παιδιά i και j.
[6.]Αν οι ρίζες είναι ταυτόσημες επιστρέφεται η τιμή του $M[m,n]$ του πίνακα αυξημένη κατά ένα. Η μεταβλητή m δηλώνει το πλήθος των κόμβων-παιδιών της ρίζας του ενός δέντρου και η μεταβλητή n το πλήθος των κόμβων-παιδιών της ρίζας του άλλου δέντρου. Σε περίπτωση που οι ρίζες δεν είναι ταυτόσημες επιστρέφεται το ίδιο στοιχείο του πίνακα χωρίς να προστεθεί σε αυτό η μονάδα.

Μια άλλη μέθοδος σύγκρισης που εφαρμόζεται από συστήματα που αναπαριστούν τα αρχεία πηγαίου κώδικα ως αφηρημένα συντακτικά δέντρα είναι ο πυρήνας δέντρου [20]. Η ομοιότητα μεταξύ δύο συντακτικών δέντρων T_1 και T_2 μπορεί να μετρηθεί από τον πυρήνα του δέντρου και ορίζεται ως:

$$K(T_1, T_2) = h(T_1) \cdot h(T_2) = \sum_{s_i \in (S_{T1} \cup S_{T2})} cnt(s_i, T_1) cnt(s_i, T_2) \tag{3.5}$$

όπου $h(T) = [h_1(T), h_2(T), \cdots, h_n(T)]$ και $h_i(T)$ ο αριθμός εμφάνισης του υποδέντρου i στο T. Επίσης, με S_T συμβολίζεται το σύνολο όλων των υποδέντρων του T. Όταν υπολογίζεται άμεσα το $K(T_1, T_2)$ πρέπει να απαριθμηθεί κάθε υποδέντρο των T_1 και T_2. Στη συνέχεια υπολογίζονται τα $cnt(s_i, T_1)$ και $cnt(s_i, T_2)$ αντίστοιχα. Μετά τον υπολογισμό του πυρήνα ανάμεσα στα δύο δέντρα απαιτείται κανονικοποίηση, με τον τύπο της συνημιτονικής ομοιότητας (σχέση (2.5)) για τα διανύσματα $h(T_1)$ και $h(T_2)$ που με τη μορφή του πυρήνα δέντρων παίρνει τη μορφή της ακόλουθης εξίσωσης:

$$cos\_K(T_1, T_2) = \frac{K(T_1, T_2)}{\sqrt{K(T_1, T_1) K(T_2, T_2)}} \tag{3.6}$$

Ένα σύστημα μπορεί να λαμβάνει υπόψη μόνο τη δομή χωρίς κάποια άλλη πληροφορία για τις εντολές. Αυτή η τακτική ακολουθείται συχνά από συστήματα που βασίζονται σε γράφους κλήσης συναρτήσεων [19]. Ο γράφος κλήσης συναρτήσεων από το αρχείο πηγαίου κώδικα διασπάται σε απλούστερους γράφους με συγκεκριμένο αριθμό κορυφών. Στη συνέχεια, μετράται το πλήθος των υπογράφων που προέκυψαν για κάθε δυνατό τύπο υπογράφου που μπορεί να δημιουργηθεί βάσει του αριθμού των κορυφών που έχει επιλεγεί. Έτσι, κάθε αρχείο πηγαίου κώδικα P εκφράζεται με ένα σύνολο της μορφής $S_m(P) = \{T(P), MFD(P)\}$. Η μεταβλητή $T(P)$ υποδηλώνει την κάθε δυνατή μορφή του υπογράφου και η μεταβλητή $MFD(P)$ υποδηλώνει το ποσοστό εμφάνισης του υπογράφου αυτού σε ολόκληρο το σύνολο των υπογράφων που προέκυψε από το αρχείο πηγαίου κώδικα. Η τιμή ομοιότητας σε αυτή την περίπτωση δίνεται από τον τύπο:

$$Hs(P, D) = \frac{\sum_{i=1}^{n} |MFD_r - 1|}{n}$$

$$MFDr = \frac{MFD(P_i)}{MFD(D_i)}$$

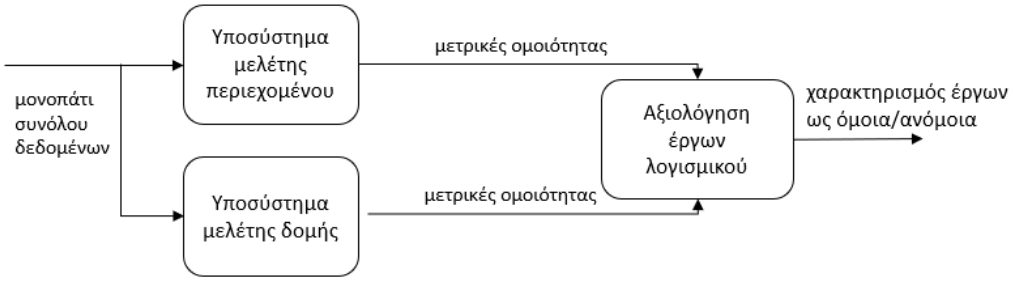
όπου n το σύνολο των διαφορετικών τύπων υπογράφου. Στον αριθμητή βρίσκεται εκείνο το αρχείο πηγαίου κώδικα όπου το ποσοστό του πρώτου υπογράφου είναι μεγαλύτερο σε σχέση με το δεύτερο. Επίσης, για αυτό τον αλγόριθμο μικρή τιμή ομοιότητας συνεπάγεται μεγάλη ομοιότητα μεταξύ των δύο αρχείων.

Τέλος, εφόσον έχει πραγματοποιηθεί η σύγκριση όλων των αρχείων πηγαίου κώδικα, ορίζεται μια τιμή αποκοπής. Η τιμή αυτή διαφέρει από σύστημα σε σύστημα. Η τιμή αποκοπής είναι ανάλογη του εύρους των τιμών ομοιότητας που παράγονται κατά τη σύγκριση. Αν η τιμή ομοιότητας μια σύγκρισης δύο αρχείων πηγαίου κώδικα είναι μεγαλύτερη από την τιμή αποκοπής τότε αυτά θεωρούνται όμοια. Σε αντίθετη περίπτωση θεωρούνται ανόμοια.

4 Υλοποίηση

Στο κεφάλαιο αυτό αναλύεται το σύστημα που υλοποιήσαμε. Επίσης αναλύονται όλα τα υποσυστήματα από τα οποία αποτελείται και περιγράφεται η δομή τους και ο τρόπος λειτουργίας τους.

Στο σύστημά μας συγκρίνουμε έργα λογισμικού σε δύο διαστάσεις: με βάση το περιεχόμενο των αρχείων πηγαίου κώδικα από τα οποία αποτελούνται και τη δομή τους. Με τη μελέτη του περιεχομένου ερευνούμε τις διαφορές που έχουν δύο έργα λογισμικού από άποψη κειμένου και δε λαμβάνουμ υπόψη δομικά τους χαρακτηριστικά. Αντιθέτως, κατά τη μελέτη της δομής εξετάζουμε κατά πόσο είναι παρόμοια η αλληλουχία κλήσης των συναρτήσεων στα δύο έργα λογισμικού, η οποία και θεωρούμε ότι χαρακτηρίζει μοναδικά κάθε έργο λογισμικού. Με τη μελέτη αυτών των δύο στοιχείων μπορούμε να αποκτήσουμε μια σφαιρική εικόνα για τα σημεία ομοιότητας δύο έργων λογισμικού.

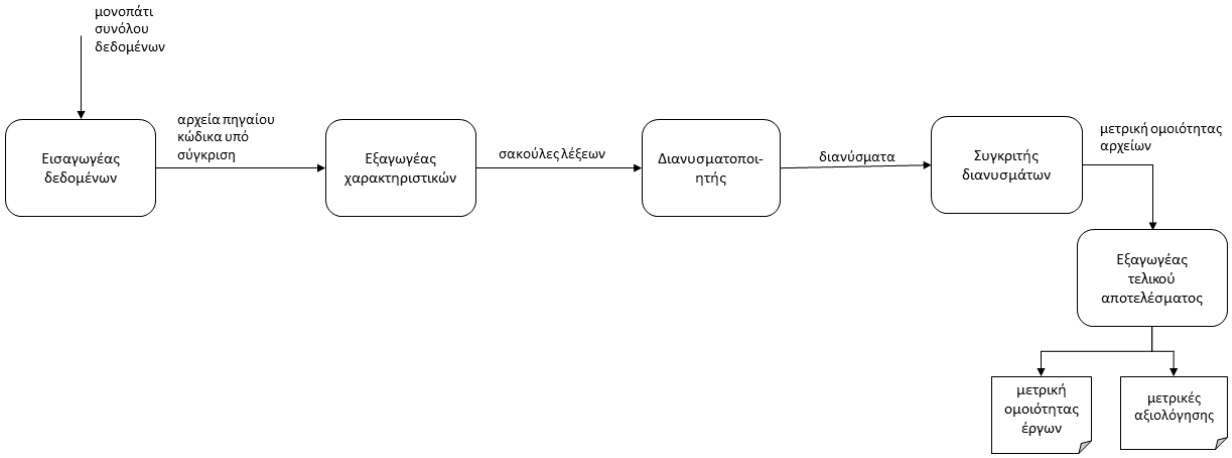


Σχήμα 3: Υλοποιημένο σύστημα

4.1 Υποσύστημα μελέτης περιεχομένου πηγαίου κώδικα

Αρχικά, μελετάμε το περιεχόμενο του πηγαίου κώδικα. Σκοπός της μελέτης αυτής είναι η έρευνα του λεξιλογίου, των τύπων μεταβλητών και των λέξεων-κλειδιών της γλώσσας προγραμματισμού που χρησιμοποιήθηκε. Ένας προγραμματιστής που έχει αντιγράψει πηγαίο κώδικα ενός άλλου έργου λογισμικού, συνήθως δεν προχωρά στη ριζική μεταβολή των ονομάτων των μεταβλητών. Ακόμα και όταν συμβαίνει αυτό, οι λέξεις-κλειδιά της γλώσσας προγραμματισμού που χρησιμοποιούνται (όπως οι λέξεις για να ορίσουν έναν βρόχο επανάληψης) και οι τύποι των μεταβλητών παραμένουν αναλλοίωτοι. Με βάση αυτές τις παραδοχές συχνά χρησιμοποιούνται μέθοδοι που αναλύουν τον πηγαίο κώδικα ως απλό κείμενο [16, 18].

Κάθε έργο λογισμικού αποτελείται από αρχεία πηγαίου κώδικα, τα οποία και πρέπει να συγκριθούν μεταξύ τους. Αφότου υπολογιστεί η ομοιότητα μεταξύ των αρχείων μπορεί να υπολογιστεί και η ομοιότητα των έργων.



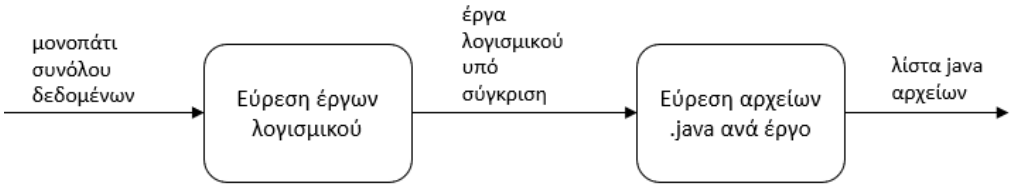
Σχήμα 4: Υποσύστημα μελέτης περιεχομένου πηγαίου κώδικα

Το υποσύστημα, όπως παρουσιάζεται και στο Σχήμα 4, αποτελείται από τον εισαγωγέα δεδομένων, τον εξαγωγέα χαρακτηριστικών, το διανυσματοποιητή, το συγκριτή διανυσμάτων και και τον εξαγωγέα μετρικών. Ο εισαγωγέας δεδομένων δέχεται ως είσοδο το μονοπάτι που βρίσκεται αποθηκευμένο το σύνολο

δεδομένων τοπικά και εξάγει τα μονοπάτια των αρχείων πηγαίου κώδικα για κάθε έργο λογισμικού. Ο εξαγωγέας λαμβάνει ως είσοδο το μονοπάτι (path) στο οποίο βρίσκεται αποθηκευμένο ένα αρχείο πηγαίου κώδικα. Ο εξαγωγέας χαρακτηριστικών επεξεργάζεται τα αρχεία και παράγει σακούλες λέξεων, μία για το κάθε αρχείο. Στη συνέχεια, ο διανυσματοποιητής δέχεται ως είσοδο μια λίστα από σακούλες λέξεων και τις μετατρέπει σε διανύσματα. Κάθε διάνυσμα αναπαριστά το αρχείο πηγαίου κώδικα στον επιλεγμένο διανυσματικό χώρο. Ο συγκριτής διανυσμάτων δέχεται ως είσοδο δύο διανύσματα και τα συγκρίνει. Το αποτέλεσμα αυτού είναι μια τιμή ομοιότητας των δύο αρχείων. Τέλος, ο εξαγωγέας μετρικών δέχεται τις τιμές ομοιότητας των αρχείων δύο έργων λογισμικού και αναλαμβάνει την αντιστοίχιση αυτών σε ζευγάρια, τον υπολογισμό της συνολικής ομοιότητας δύο έργων λογισμικού και την αξιολόγηση των αποτελεσμάτων με τη χρήση διάφορων μετρικών.

4.1.1 Εισαγωγέας δεδομένων

Ο εισαγωγέας δεδομένων (βλπ. Σχήμα 5) είναι ένα εργαλείο που αναλαμβάνει την ορθή και αυτόματη λειτουργία του υποσυστήματος μελέτης περιεχομένου. Είναι αρμοδιότητά του η σύγκριση όλων των έργων λογισμικού του συνόλου δεδομένων μεταξύ τους. Κατά τη σύγκριση των έργων λογισμικού χρησιμοποιώντας κάποια μέθοδο σύγκρισης διανυσμάτων ακολουθείται η εξής διαδικασία: ο εισαγωγέας δημιουργεί έναν πίνακα κατακερματισμού που έχει ως κλειδί το μονοπάτι ενός φακέλου και ως τιμή μια λίστα με το σύνολο των έργων και για κάθε έργο δημιουργείται και μια λίστα με αρχεία πηγαίου κώδικα που περιέχονται σάυτό και πρέπει να τεθούν σε σύγκριση.



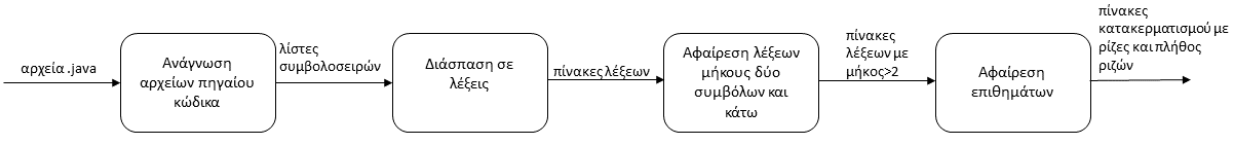
Σχήμα 5: Εισαγωγέας δεδομένων

Η ομαλή εκτέλεση του αλγορίθμου λανθάνουσας σημασιολογικής ανάλυσης πραγματοποιείται με τις μεθόδους της κλάσης LSAComparison. Ο πίνακας κατακερματισμού δημιουργείται με της βοήθεια της μεθόδου mapFiles.

Η εκτέλεση του αλγορίθμου που χρησιμοποιεί κάποια μέθοδο σύγκρισης διανυσμάτων υλοποιείται με τις κλάσεις FileTokenComparison και FunctionTokenComparison. Ο πίνακας κατακερματισμού δημιουργείται με της βοήθεια της μεθόδου mapFiles. Η λίστα με τα αρχεία πηγαίου κώδικα προς σύγκριση μιας εφαρμογής παράγεται με τη μέθοδο listFilesForFolder.

4.1.2 Εξαγωγέας χαρακτηριστικών

Ο εξαγωγέας (βλπ. Σχήμα 6) λαμβάνει ως είσοδο ένα μονοπάτι. Το μονοπάτι δείχνει την τοποθεσία στην οποία είναι αποθηκευμένο το υπό σύγκριση αρχείο πηγαίου κώδικα μιας κλάσης. Το εργαλείο αναλαμβάνει την ανάγνωση πηγαίου κώδικα, την αφαίρεση της περιττής πληροφορίας, τη διάσπαση του κώδικα σε λέξεις και τη θεματοποίηση των λέξεων. Έτσι, ο υπό σύγκριση πηγαίος κώδικας προετοιμάζεται κατάλληλα ώστε να μπορεί να μετατραπεί σε επόμενο βήμα σε διάνυσμα.



Σχήμα 6: Εξαγωγέας χαρακτηριστικών

Το πρώτο βήμα αυτού του εργαλείου είναι η εισαγωγή του πηγαίου κώδικα. Επομένως, μια μέθοδος (importfile) αναλαμβάνει την ανάγνωση του πηγαίου κώδικα ανά γραμμή και την αποθήκευσή του σε μια λίστα. Κατά την εισαγωγή του κώδικα στο σύστημα τα σχόλια μπορούν να αφαιρεθούν. Αυτό συμβαίνει καθώς πολλές μέθοδοι της βιβλιογραφίας πραγματοποιούν συγκρίσεις μεταξύ έργων λογισμικού χωρίς την

παρουσία σχολίων. Επόμένως, είναι απαραίτητη η δυνατότητα αφαίρεσης των σχολίων ώστε να γίνει μια μελέτη σε επόμενο στάδιο κατά πόσο αυτά συνεισφέρουν στην τελική ομοιότητα των έργων. Η μέθοδος replaceComments εξετάζει κάθε γραμμή του κώδικα. Αντικαθιστά τα σχόλια με κενό χαρακτήρα και επιστρέφει τον υπόλοιπο κώδικα.

Δύο έργα λογισμικού που είναι όμοια είναι πολυ πιθανό να έχουν και όμοιες μεθόδους. Κατά τον υπολογισμό ομοιότητας ενδιαφερόμαστε για την ομοιότητα των μεθόδων έργου λογισμικού. Ωστόσο, δύο όμοια έργα λογισμικού δε σημαίνει ότι θα έχουν απαραίτητα τις ίδιες μεθόδους στις ίδιες κλάσεις. Επομένως, απαιτείται η μελέτη και σε επίπεδο συναρτήσεων. Το σύστημα, έτσι, υποστηρίζει και το διαχωρισμό του κώδικα σε συναρτήσεις. Ο διαχωρισμός αυτός γίνεται με τη μέθοδο importFunctions, εφόσον πρώτα αφαιρεθούν τα σχόλια με τη μέθοδο replaceComments. Αυτό συμβαίνει ώστε να αποφευχθεί η λανθασμένη θεώρηση κάποιων γραμμών κώδικα που περικλείονται με τα σύμβολα των σχολίων ως συνάρτηση.

Μετά την εισαγωγή του πηγαίου κώδικα είτε με τη μορφή αρχείων είτε με τη μορφή συνάρτησης, κάθε πρόταση διασπάται σε λέξεις. Η διάσπαση υλοποιείται με τη μέθοδο splitSentenceWords. Η διάσπαση πραγματοποιείται από σύμβολο ή κενό χαρακτήρα σε σύμβολο ή κενό χαρακτήρα. Κανένα σύνολο διαδοχικών συμβόλων δε λαμβάνεται υπόψη ως λέξη. Επίσης, γίνεται διαχωρισμός λέξεων τύπου camelCase σε camel και case. Ακόμη, τα κεφαλαία γράμματα μετατρέπονται σε μικρά. Ο λόγος που συμβαίνει αυτό είναι επειδή οι γλώσσες προγραμματισμού δε θεωρούν τα κεφαλαία γράμματα με τα αντίστοιχα μικρά ότι είναι τα ίδια σύμβολα. Όλες οι μέθοδοι που αναφέρθηκαν ανήκουν στην κλάση ImportedFile.

Ακόμη, λαμβάνονται υπόψη μόνο οι όροι όπου έχουν περισσότερους από δύο χαρακτήρες. Οποιοσδήποτε άλλος όρος θεωρείται ότι δεν περιέχει κάποια πληροφορία. Επομένως, θα αποτελούσε θόρυβο στο σύστημά μας. Σε αυτή την κατηγορία ανήκουν κυρίως ονόματα μεταβλητών που συμμετέχουν σε διεργασίες ρουτίνας. Για παράδειγμα, για τις μεταβλητές που έχουν το ρόλο του μετρητή σε έναν βρόχο επανάληψης *for* χρησιμοποιούνται συνήθως συντομα ονόματα (π.χ. *i* ή *j*) που δεν μπορούν να συνδράμουν στον υπολογισμό ομοιότητας, ενώ για τις μεταβλητές που υπάρχουν μέσα σε ένα αρχείο και αποθηκεύουν σημαντική πληροφορία χρησιμοποιούνται συνήθως πιο περιγραφικά ονόματα τα οποία μπορούν να έχουν καθοριστικό ρόλο στον υπολογισμό ομοιότητας. Στον Πίνακα 1 παρουσιάζονται ορισμένα παραδείγματα φράσεων ώστε να γίνει πιο κατανοητός ο τρόπος αφαίρεσης των λέξεων.

Όροι πριν την απομάκρυνη	Όροι μετά την απομάκρυνση
you, are, super	you, are, super
delete, an, object	delete, object
add, a, number	add, number

Πίνακας 1: Παράδειγμα απομάκρυνσης λέξεων μικρού μήκους

Στη συνέχεια, απαιτείται η θεματοποίηση των όρων. Η αφαίρεση αυτών πραγματοποιείτα με τον αλγόριθμο θεματοποίησης Lovin, ο οποίος παρήγαγε καλύτερα αποτελέσματα σε σχέση με τον αλγόρθμο Porter. Ο αλγόριθμος αυτός αρχικά μετρά το μέγεθος της λέξης. Ανάλογα με αυτό, ελέγχεται αν μία λέξη περιέχει ένα συγκεκριμένο επίθημα. Στη συνέχεια, ο υπόλοιπος όρος ελέγχεται από 34 διαφορετικούς κανόνες για να λάβει τη μορφή ρίζας. Οι μέθοδοι της κλάσης LovinStemmer υλοποιούν αυτή τη λειτουργία. Συχνά η τελική μορφή μιας λέξης αφότου θεματοποιηθεί μπορεί να μην είναι υπαρκτή. Αυτό δε μας ενοχλεί, καθώς επεξεργάζεται με την ίδια διαδικασία όλες τις λέξεις και δεν επιδρά αρνητικά στον τελικό υπολογισμό της ομοιότητας. Στον Πίνακα 2 φαίνονται μερικά παραδείγματα όρων για τους οποίους πραγματοποιείται θεματοποίηση.

Όροι πριν την θεματοποίηση	Οροι μετά τη θεματοποίηση
doing, some, testing	do, some, test
inflectional, rules	inflect, rule
added, 2, numbers	ad, 2, number

Πίνακας 2: Παράδειγμα θεματοποίησης όρων

Για παράδειγμα, στον Πίνακα 3 παρουσιάζεται το αποτέλεσμα της παραπάνω διαδικασία για δύο αρχεία προς σύγκριση.

Αρχείο	Περιεχόμενο	Σακούλα λέξεων
d1	bag of words of	bag=1, of=2, word=1
d2	items bag	item=1, bag=1

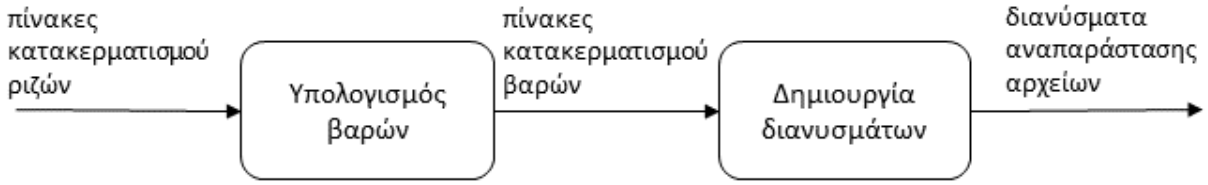
Πίνακας 3: Μετατροπή αρχείων σε σακούλες λέξεων

4.1.3 Διανυσματοποιητής

Το εργαλείο αυτό μετατρέπει τη σακούλα λέξεων σε διάνυσμα. Αρχικά, δέχεται ως είσοδο αντικείμενα τύπου ImportedFile. Τα αντικείμενα αυτά ισοδυναμούν με τα αρχεία πηγαίου κώδικα δύο διαφορετικών έργων λογισμικού. Αυτά θα πρέπει να συγκριθούν μεταξύ τους. Δημιουργείται, στη συνέχεια, μια μεγάλη σακούλα λέξεων. Η σακούλα αυτή περιλαμβάνει όλους τους διαφορετικούς όρους από τις σακούλες όλων των αρχείων. Αυτό γίνεται ώστε να δημιουργηθεί μια λίστα όρων όπου σε κάθε όρο θα αντιστοιχεί ένας μοναδικός αριθμός. Το μέγεθος της λίστας θα ισούται με τον αριθμό των όρων του διανύσματος. Ο μοναδικός αριθμός κάθε όρου υποδηλώνει τη θέση που αντιστοιχεί ο όρος αυτός στο διάνυσμα. Για τις σακούλες λέξεων που αναφέραμε προηγουμένως, θα προκύψει η σακούλα που παρουσιάζεται στον Πίνακα 4.

Όρος	Μοναδικός αριθμός κάθε όρου
bag	0
of	1
word	2
item	3

Πίνακας 4: Παράδειγμα πίνακα θέσης όρων



Σχήμα 7: Διανυσματοποιητής

Για κάθε αρχείο γίνεται η καταμέτρηση των τελικών ριζών και δημιουργείται μια λίστα με τις διαφορετικές ρίζες που υπάρχουν σε αυτό. Για καθεμιά από αυτές αναγράφεται ο αριθμός εμφάνισής τους στη σακούλα λέξεων. Αυτη η διαδικασία διευκολύνει τη δημιουργία του διανύσματος. Το σύστημα υποστηρίζει δύο είδη αναπαραστάσεων: η πρώτη εξυπηρετεί τον υπολογισμό ομοιότητας και βασίζεται στη σύγκριση διανυσμάτων, ενώ η δεύτερη τον αλγόριθμο λανθάνουσας σημασιολογικής ανάλυσης. Για την πρώτη επιλέγεται η αναπάρασταση συχνότητας όρου-αντίστροφης συχνότητας αρχείου. Με την αναπαράσταση αυτή τα διανύσματα αντιπροσωπεύουν με μεγαλύτερη ακρίβεια της σακούλες λέξεων και συνεπώς τα αρχεία πηγαίου κώδικα. Έτσι, η τιμή ομοιότητας που θα προκύψει σε επόμενο βήμα θα είναι πιο ρεαλιστική και αξιόπιστη. Η συχνότητα tf όρου i σε ένα έγγραφο d δίνεται από τη σχέση:

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{4.1}$$

όπου η μεταβλητή $f_{t,d}$ συμβολίζει τον αριθμό εμφάνισης του όρου t στο έγγραφο d. Η αντίστροφη συχνότητα αρχείου δίνεται από τη σχέση:

$$idf(t) = 1 + \log \frac{N}{n_i} \tag{4.2}$$

όπου t είναι ο όρος για τον οποίο υπολογίζεται η αντίστροφη συχνότητα, N το σύνολο των υπο μελέτη αρχείων και n_i ο αριθμός των αρχείων που περιέχουν τον όρο t. Έτσι, σε κάθε όρο i του διανύσματος που αποτελεί τον όρο t του αρχείου πηγαίου κώδικα αντιστοιχεί το γινόμενο $tf(t,d) \cdot idf(t)$.Για τη

συχνότητα όρου χρησιμοποιείται ουσιαστικά το ποσοστό επί τοις εκατό της συχνότητας εμφάνισης σε ένα αρχείο. Έτσι, η τιμή αυτή είναι κανονικοποιημένη. Για την αντίστροφη συχνότητα αρχείου η συνάρτηση είναι λογαριθμική επειδή είναι επιθυμητό όροι που συμμετέχουν σε πολλά αρχεία να αποκτούν μικρή βαρύτητα και όροι που συμμετέχουν σε λίγα, μεγαλύτερη. Επίσης, ο λογάριθμος εξομαλύνει τη μεγάλη τιμή του κλάσματος. Η μονάδα προστίθεται ώστε να χαθεί η πληροφορία ενός όρου που εμφανίζεται σε όλα τα αρχεία. Για παράδειγμα, αν δύο έγγραφα περιέχουν έναν όρο με συχνότητα 0.9 και 0.2, στην περίπτωση της μη ύπαρξης της μονάδας η τελική συχνότητα θα μηδενιστεί. Τα διανύσματα των υπο σύγκριση αρχείων εξάγονται από το παρόν εργαλείο. Τα διανύσματα που θα προκύψουν για κάθε αρχείο φαίνονται στον Πίνακα 4.1.3.

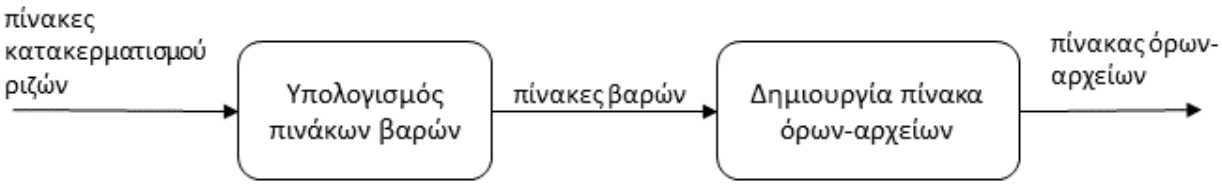
Θέση όρων	d1 χωρίς βάρη	d2 χωρίς βάρη	d1 με βάρη	d2 με βάρη
0	1	1	0.25	0.5
1	2	0	0.847	0
2	1	0	0.423	0
3	0	1	0	0.847

Πίνακας 5: Διανύσματα

Για τη λανθάνουσα σημασιολογική ανάλυση είναι διαφορετική η διαδικασία της διανυσματοποίησης. Επομένως, χρησιμοποιείται διαφορετικό εργαλείο υλοποίησης. Ακολουθώντας τη διαδικασία που αναφέρθηκε στο Υποκεφάλαιο 2.1 δημιουργείται ένας πίνακας A όρων-αρχείων. Για τη διεκπεραίωση των διαδικασιών αυτών χρησιμοποιείται η κλάση SpecialMatrices. Αρχικά, δημιουργείται ένας πίνακας συχνότητας όρου με τη μέθοδο createMatrixWithTerm. Κάθε στοιχείο f_{ij} του πίνακα αυτού δείχνει τον αριθμό εμφάνισης του όρου που αντιστοιχεί στη θέση i στο αρχείο j. Στη συνέχεια, δημιουργούνται δύο πίνακες: ένας πίνακας γενικού βάρους και ένας πίνακας συνημιτονικής κανονικοποίησης. Κάθε στοιχείο του πίνακα γενικού βάρους ισούται με το κανονικό γενικό βάρος του όρου i που δίνεται από τη σχέση (2.16β'). Κάθε στοιχείο του πίνακα συνημιτονικής κανονικοποίησης δίνεται από τον τύπο (2.1γ'). Με τα βάρη αυτά, τα διανύσματα αφενός δίνουν μεγαλύτερη αξία σε όρους που έχουν μήκος ίσο με τη μονάδα Οι μέθοδοι υλοποίησης αυτών την πινάκων είναι οι createNormalMatrix και createCosineMatrix. Στη συνέχεια οι τρεις αυτοί πίνακες πολλαπλασιάζονται στοιχείο προς στοιχείο για να δημιουργήσουν το σταθμισμένο πίνακα A όρων-αρχείων. Ο πολλαπλασιασμός υλοποιείται με τη μέθοδο createFinalMatrix. Κάθε διάνυσμα αρχείου έχει μήκος ίσο με τη μονάδα. Έτσι οι πίνακες που προκύπτουν είναι οι εξής:

	d1	d2
bag	0.447	0.577
of	0.632	0
word	0.632	0
item	0	0.816

Πίνακας 6: LSA Διανύσματα

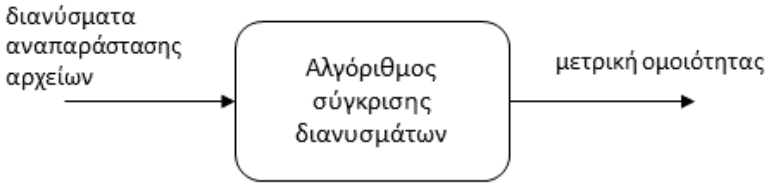


Σχήμα 8: Διανυσματοποιητής λανθάνουσας σημασιολογικής ανάλυσης

4.1.4 Συγκριτής διανυσμάτων

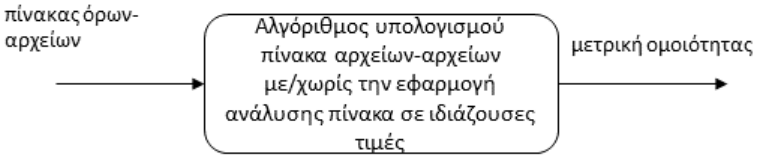
Αφότου παράξουμε τα διανύσματα με το εργαλείο που αναλύθηκε προηγουμένως, απαιτείται η σύγκριση αυτών. Χρησιμοποιούνται δύο είδη συγκριτή διανυσμάτων: ένας που εφαρμόζει τον αλγόριθμο του υπολογισμού ομοιότητας δύο αρχείων πηγαίου κώδικα χρησιμοποιώντας τεχνικές σύγκρισης διανυσμάτων και ένας που εφαρμόζει τον αλγόριθμο της λανθάνουσας σημασιολογικής ανάλυσης.

Ο πρώτος συγκριτής λαμβάνει δύο διανύσματα αρχείων πηγαίου κώδικα. Ο συγκριτής υπολογίζει τη συνημιτονική ομοιότητα των δύο διανυσμάτων βάσει του τύπου (2.5). Πρόκειται για μια μέθοδο που χρησιμοποιείται κατά κύριο λόγο σε τέτοιου είδους συστήματα και με αυτόν τον τρόπο μετράται η γωνία των διανυσμάτων. Παράλληλα διανύσματα, συνεπάγονται όμοια αρχεία, ενώ τα κάθετα είναι ανόμοια. Η τιμή, επομένως, που εξάγεται είναι μεταξύ μηδέν και ένα. Η τιμή είναι ένα, αν τα διανύσματα είναι πανομοιότυπα ενώ είναι μηδέν, αν τα αρχεία είναι εντελώς ανόμοια.



Σχήμα 9: Συγκριτής διανυσμάτων

Ο δεύτερος συγκριτής αρχικά υπολογίζει τον πίνακα αρχείων-αρχείων. Ο πίνακας αυτός υπολογίζεται πολλαπλασιάζοντας τον πίνακα A με τον ανάστροφό του. Αυτό επιτρέπεται εφόσον όλα τα διανύσματα είναι κανονικοποιημένα και έχουν μέτρο ίσο με τη μονάδα. Επομένως, οι τιμές ομοιότητας που θα προκύψουν είναι απόλυτα συγκρίσιμες μεταξύ τους.



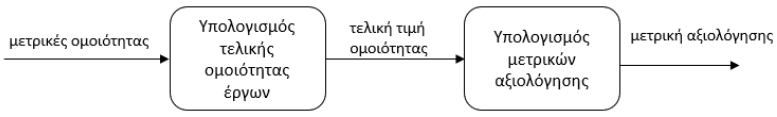
Σχήμα 10: Συγκριτής λανθάνουσας σημασιολογικής ανάλυσης

Ο συγκριτής επίσης, εφαρμόζει και τον αλγόριθμο ανάλυσης σε ιδιάζουσες τιμές. Οι σχέσεις μεταξύ των όρων δε διαμορφώνονται ρητά στη δημιουργία του χώρου της λανθάνουσας σημασιολογικής ανάλυσης και αυτό καθιστά δύσκολο να κατανοηθεί ο τρόπος λειτουργίας της γενικά. Παρέχουν μαθηματική απόδειξη ότι ο αλγόριθμος SVD ενσωματώνει τη συν-εμφάνιση και από αυτό η λανθάνουσα σημασιολογική ανάλυση παράγει σημασιολογική γνώση του κειμένου. Η ανάλυση του σταθμισμένου πίνακα στους τρεις πίνακες που πραγματοποιείται σύμφωνα με την περιγραφή της υποενότητας 2.1 . Έτσι, ο συγκριτής παράγει και έναν πίνακα A βαθμού k. Στη συνέχεια υπολογίζεται και πάλι ο πίνακας αρχείων-αρχείων. Ο πίνακας δείχνει διαφορετικές τιμές ομοιότητας μεταξύ των αρχείων σε σχέση με τον αντίστοιχο πίνακα στον οποίο δεν εφαρμόστηκε ο αλγόριθμος ανάλυσης σε ιδιάζουσες τιμές.

Η αποσύνθεση του πίνακα A πραγματοποιείται με τις μεθόδους της κλάσης SVD η οποία χρησιμοποιεί τη βιβλιοθήκη Jama[1].

4.1.5 Εξαγωγέας μετρικών

Ο εξαγωγέας μετρικών (βλπ. Σχήμα11) είναι ένα εργαλείο που αναλαμβάνει τον υπολογισμό της τελικής ομοιότητας δύο έργων λογισμικού. Επιπροσθέτως, υπολογίζει τις διάφορες μετρικές αξιολόγησης του υποσυστήματος χρησιμοποιώντας τις τιμές ομοιότητας που προκύπτουν από τις συγκρίσεις όλων των έργων.



Σχήμα 11: Εξαγωγέας μετρικών

[1] https://math.nist.gov/javanumerics/jama

Κατά τη σύγκριση δύο έργων με τη μέθοδο της λανθάνουσας σημασιολογικής ανάλυσης δίνονται όλα τα αρχεία πηγαίου κώδικα και από τα δύο έργα λογισμικού ως είσοδο στον εξαγωγέα. Έτσι, με τη διαδικασία που περιγράψαμε, παράγεται ο πίνακας αρχείων-αρχείων. Ο εξαγωγέας μετρικών πραγματοποιεί την αντιστοίχιση των αρχείων πηγαίου κώδικα της μιας εφαρμογής με αρχεία της άλλης. Για να πραγματοποιηθεί αυτό, αναζητάται το ζευγάρι αρχείων που έχει τη μέγιστη ομοιότητα. Η τιμή ομοιότητας αποθηκεύεται σε μία μεταβλητή. Στη συνέχεια, αναζητάται και πάλι το ζευγάρι με την μέγιστη ομοιότητα. Τα αρχεία που έχουν αντιστοιχηθεί ήδη, εξαιρούνται από τη νέα αναζήτηση. Η μέγιστη τιμή κρατάται και πάλι σε κάποια μεταβλητή. Η διαδικασία αυτή επαναλαμβάνεται εώς ότου τουλάχιστον το ένα από τα δύο υπό σύγκριση έργα λογισμικού αντιστοιχήσει τα αρχεία πηγαίου κώδικα με αρχεία του άλλου έργου. Ως συνολική τιμή ομοιότητας θεωρείται ο μέσος όρος των τιμών ομοιότητας των ζευγαριών που αντιστοιχήθηκαν.

Κατά τη σύγκριση αρχείων με τη μέθοδο σύγκρισης διανυσμάτων, κάθε αρχείο πηγαίου κώδικα ενός έργου λογισμικού συγκρίνεται με κάθε αρχείο ενός άλλου έργου. Για κάθε σύγκριση παράγεται μια τιμή ομοιότητας βάσει του προηγούμενου βήματος. Από αυτή τη σύγκριση κρατούνται οι τιμές ομοιότητας και τα αρχεία πηγαίου κώδικα για τα οποία αντιστοιχεί η τιμή αυτή. Από τις τιμές αυτές αναζητάται η μέγιστη τιμή. Στη συνέχεια, εξαιρούνται τα αρχεία αυτά για τα οποία η τιμή ομοιότητας είναι μέγιστη καθώς και οι τιμές ομοιότητας που προέρχονται από συγκρίσεις αυτών των αρχείων με άλλα. Από τις εναπομείναντες τιμές, αναζητάται ξανά η μέγιστη τιμή ομοιότητας. Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου όλα τα αρχεία πηγαίου κώδικα τουλάχιστον του ενός έργου λογισμικού αντιστοιχηθούν με αρχεία του άλλου. Η τιμή ομοιότητας των αρχείων λογισμικού ισούται με το μέσο όρο των τιμών ομοιότητας των αρχείων που αντιστοιχήθηκαν μεταξύ τους. Εφόσον πραγματοποιηθεί το σύνολο των συγκρίσεων, υπολογίζεται μια μετρική αξιολόγησης των αλγορίθμων. Η μετρική που χρησιμοποιείται θα αναλυθεί στο επόμενο κεφάλαιο.

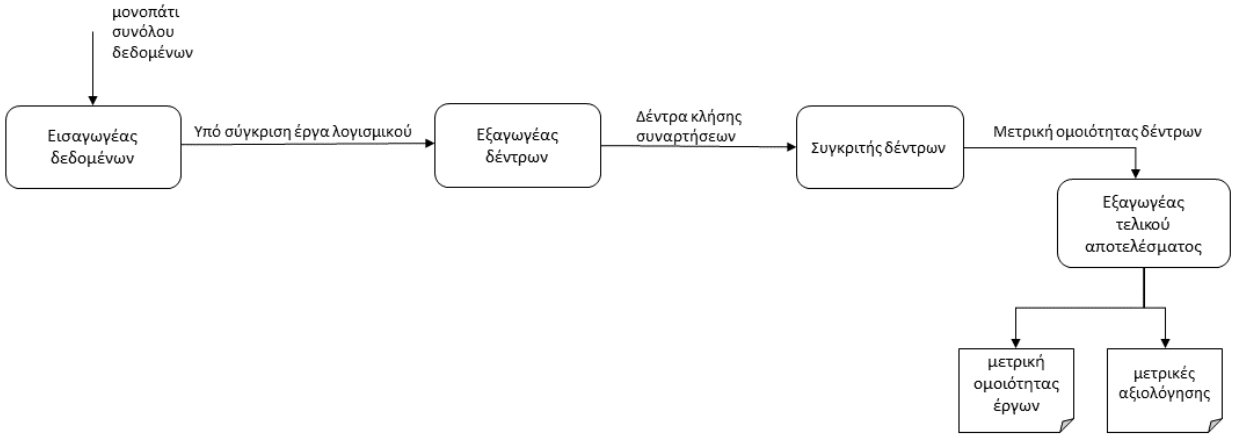
Για την εφαρμογή του αλγορίθμου λανθάνουσας σημασιολογικής ανάλυσης χρησιμοποιούνται μέθοδοι της κλάσης LSAComparison. Οι μέθοδοι compareFilesOfSameFolder και compareFilesOfDifferentFolder συγκρίνουν έργα που ανήκουν στον ίδιο και σε διαφορετικό φάκελο αντίστοιχα. Αυτές οι μέθοδοι χρησιμοποιούν τη μέθοδο executeTwoFiles που συγκρίνει δύο έργα λογισμικού μεταξύ τους. Η μέθοδος findMaxValue αναζητά τη μέγιστη τιμή ομοιότητας των αρχείων ώστε να γίνει αντιστοίχιση μεταξύ τους και επιστρέφει την τιμή ομοιότητας των έργων βάσει αυτής της αντιστοίχισης.

Για την εφαρμογή του αλγορίθμου σύγκρισης διανυσμάτων χρησιμοποιούνται μέθοδοι των κλάσεων FileTokenComparison και FunctionTokenComparison. Οι μέθοδοι compareFilesOfSameFolder και compareFilesOfDifferentFolder συγκρίνουν έργα λογισμικού που ανήκουν στον ίδιο και σε διαφορετικό φάκελο αντίστοιχα. Αυτό γίνεται με τη βοήθεια της μεθόδου executeTwoFiles που συγκρίνει δύο έργα μεταξύ τους. Η μέθοδος compareJavaFiles συγκρίνει τα αρχεία πηγαίου κώδικα μεταξύ τους.

4.2 Υποσύστημα μελέτης δομής πηγαίου κώδικα

Μετά τη μελέτη του περιεχομένου του πηγαίου κώδικα εξετάζεται και η δομή του. Η δομή του πηγαίου κώδικα περιλαμβάνει τη διάρθρωση των μεθόδων και των εντολών που περιέχει. Συχνά παρουσιάζεται το πρόβλημα ότι αν ένα έργο λογισμικού υλοποιεί ένα κοινό πρόβλημα, υπάρχουν μέθοδοι που υλοποιούν μια βασική διεργασία και ενδεχομένως ένας αλγόριθμος σύγκρισης να οδηγηθεί σε εσφαλμένα αποτελέσματα. Για παράδειγμα, μια μέθοδος που υπολογίζει το μέγιστο ενός πίνακα δεν μπορεί να υλοποιηθεί με πολλούς διαφορετικούς τρόπους. Δύο μέθοδοι που υλοποιούν την ίδια διεργασία καλώντας πολλές άλλες μεθόδους, είτε σύνθετες είτε μη, είναι απίθανο να καλούν τον ίδιο αριθμό εξίσου όμοιων συναρτήσεων και να προέρχονται από διαφορετικό συγγραφέα. Έτσι, για την εξέταση της δομής χρησιμοποιούνται δέντρα κλήσης από τον πηγαίο κώδικα. Ο γράφος κλήσης συναρτήσεων είναι χαρακτηριστικός για το κάθε έργο λογισμικού. Το υποσύστημα μελέτης δομής(βλπ. Σχήμα12) περιλαμβάνει τα ακόλουθα εργαλεία: τον εισαγωγέα δεδομένων, τον εξαγωγέα δέντρων, τον συγκριτή δέντρων και τον εξαγωγέα μετρικών.

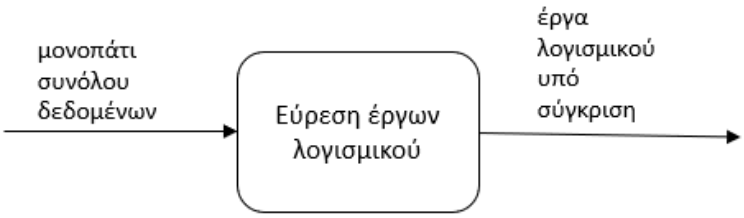
Ο εισαγωγέας δεδομένων φροντίζει είσοδο του συνόλου δεδομένων στο σύστημα καθώς και τη σύγκριση όλων των έργων μεταξύ τους. Ο εξαγωγέας δέντρων λαμβάνει ως είσοδο το μονοπάτι του έργου. Το εργαλείο αυτό εξάγει αρχικά το γράφο κλήσης δέντρων και στη συνέχει από αυτόν εξάγονται τα δέντρα κλήσης συνάρτησης. Ο συγκριτής εφαρμόζει έναν αλγόριθμο υπολογισμού ομοιότητας μεταξύ των δέντρων. Τέλος, ο εξαγωγέας μετρικών πραγματοποιεί την αντιστοίχιση μεταξύ των δέντρων ώστε να υπολογιστεί η ομοιότητα μεταξύ των έργων και εξάγει τις μετρικές αξιολόγησης που χρησιμοποιεί το σύστημα.



Σχήμα 12: Υποσύστημα μελέτης δομής του πηγαίου κώδικα

4.2.1 Εισαγωγέας δεδομένων

Ο εισαγωγέας δεδομένων (βλπ. Σχήμα 13) και σε αυτό το υποσύστημα αναλαμβάνει την ορθή λειτουργία του υποσυστήματος με τη σύγκριση όλων των έργων λογισμικού μεταξύ τους. Τροφοδοτεί το σύστημα με το σύνολο δεδομένων που πρέπει να συγκριθούν και συλλέγει τις τιμές ομοιότητας που προκύπτουν. Ο εισαγωγέας δέχεται ως είσοδο το μονοπάτι του συνόλου δεδομένων. Έτσι, δημιουργείται ένας πίνακας κατακερματισμού. Ο πίνακας αυτός περιέχει το μονοπάτι των φακέλων και λίστες με το σύνολο των έργων λογισμικού που είναι αποθηκευμένες σε αυτό.



Σχήμα 13: Εισαγωγέας δεδομένων

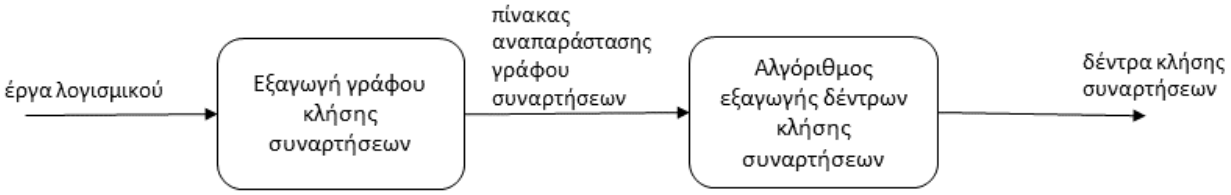
Η υλοποίηση του εισαγωγέα επιτυγχάνεται με την κλάση CallTreeComparison. Η μέθοδος mapFiles δέχεται ως όρισμα το μονοπάτι του συνόλου δεδομένων στο οποίο είναι αποθηκευμένα και εξάγει τον πίνακα κατακερματισμού που περιγράψαμε παραπάνω. Οι μέθοδοι executeFilesOfSameFolder και executeFilesOfDifferentFolder υλοποιούν συγκρίσεις έργων λογισμικού που είναι διακλαδώσεις του ίδιου και διαφορετικού έργου αντίστοιχα. Οι μέθοδοι αυτές λειτουργούν καλώντας τη μέθοδο executeComparison. Η τελευταία μέθοδος χρησιμοποιείται για τη σύγκριση δύο οποιωνδήποτε έργων.

4.2.2 Εξαγωγέας δέντρων

Ο εξαγωγέας δέντρων (βλπ. Σχήμα14) δέχεται ως είσοδο το μονοπάτι του έργου λογισμικού που θέλουμε να συγκρίνουμε. Ο εξαγωγέας αναλαμβάνει την εξαγωγή του γράφου κλήσης συνάρτησης. Ο γράφος αυτός προκύπτει από βιβλιοθήκη που επεξεργάζεται τα εκτελέσιμα αρχεία του έργου [2]. Η αναπαράστασή του γίνεται με έναν τετραγωνικό πίνακα M. Κάθε σειρά και κάθε στήλη απεικονίζει μία συνάρτηση. Αν μία συνάρτηση που αντιστοιχεί στη σειρά i καλείται από μιά συνάρτηση που αντιστοιχεί στη στήλη j τότε η τιμή του $M[i][j]$ ισούται με ένα. Η σειρά με την οποία αντιστοιχούν οι συναρτήσεις στις στήλες του πίνακα είναι ίδια με αυτή στις σειρές. Επομένως, όλα τα στοιχεία της κύριας διαγωνίου του πίνακα ισούνται με ένα. Σε κάθε άλλη περίπτωση τα στοιχεία του πίνακα είναι μηδέν.

Στη συνέχεια, προκύπτουν από το γράφο κάποια δέντρα κλήσης. Δεν εξετάζουμε όλα τα πιθανά δέντρα γιατί αφενός όσα αποτελούνται από ελάχιστους κόμβους δε φέρουν σημαντική πληροφορία για τη μέτρηση ομοιότητας του πηγαίου κώδικα και αφετέρου η υπολογιστική πολυπλοκότητα θα αυξηθεί σημαντικά. Ένα δέντρο, επομένως, πρέπει να περιέχει αρκετούς κόμβους ώστε να κριθεί ως συγκρίσιμο.

[2] `https://github.com/maniospas/JStructureAST`



Σχήμα 14: Εξαγωγέας δέντρων

Αυτό συμβαίνει διότι η πληροφορία που θα περιέχει θα πρέπει να είναι ικανοποιητική ώστε κάθε δέντρο να είναι αντιπροσωπευτικό του προγράμματος. Έτσι το αποτέλεσμα της σύγκρισης θα είναι αξιόπιστο. Η επιλογή των δέντρων γίνεται με την εξής διαδικασία:

Αλγόριθμος 7 Αλγόριθμος εξαγωγής δέντρων από γράφο κλήσης συνάρτησης

[1.] Αναζητούνται οι κόμβοι που έχουν περισσότερα από τρία παιδιά. Αυτοί οι κόμβοι αποθηκεύονται σε μία λίστα. Αν δεν υπάρχει κόμβος που να έχει περισσότερα από τρία παιδιά τότε αναζητάται ο κόμβος με τα περισσότερα παιδιά. Αυτός ο κόμβος αποτελεί τη ρίζα του δέντρου που θα εξαχθεί από αυτό το εργαλείο.

[2.] Οι κόμβοι αυτοί ταξινομούνται στη λίστα. Πρώτος είναι ο κόμβος με το μεγαλύτερο αριθμό παιδιών, ενώ τελευταίος ο κόμβος με το μικρότερο αριθμό παιδιών.

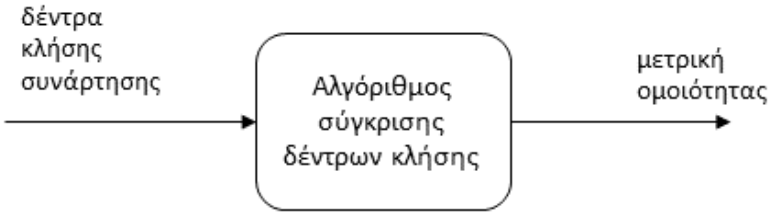
[3.] Καθένας από τους παραπάνω κόμβους θεωρείται ρίζα ενός δέντρου. Για κάθε δέντρο προκύπτει ένα σύνολο κόμβων που ανήκουν σε αυτό. Κάθε κόμβος-ρίζα επομένως συνοδεύεται από ένα σύνολο κόμβων. Για κάθε κόμβο-ρίζα της ταξινομημένης λίστας ελέγχεται αν ανήκει στο σύνολο κόμβων ενός άλλου δέντρου. Αν ανήκει, τότε ο κόμβος αυτός αφαιρείται από τη λίστα.

[4.] Εφόσον πραγματοποιηθεί έλεγχος για όλους τους κόμβους τις ταξινομημένης λίστας, οι εναπομείναντες κόμβοι αποτελούν ρίζες των δέντρων που θα αποτελούν την έξοδο του εξαγωγέα.

Η δημιουργία του πίνακα πραγματοποιείται με τη συνάρτηση generateTraversalMatrix της κλάσης ASTProject. Ο υπολογισμός των ριζών πραγματοποιείται με τη συνάρτηση calculatePositionOfRoots της κλάσης GeneratedSubTreeMatrix. Η λίστα κόμβων ενός δέντρου με δεδομένη ρίζα λαμβάνεται από τη μέθοδο getSubTreeNodePosition της ίδιας κλάσης.

4.2.3 Συγκριτής δέντρων

Ο συγκριτής δέντρων (βλπ. Σχήμα 15) υλοποιεί τους αλγορίθμους σύγκρισης που εφαρμόζονται για τον υπολογισμό ομοιότητας του πηγαίου κώδικα. Η σύγκριση γίνεται μεταξύ δύο έργων λογισμικού. Δέχεται ως είσοδο τα δέντρα που προέκυψαν από την προηγούμενη διαδικασία. Τα δέντρα συνοδεύονται με τον πηγαίο κώδικα κάθε συνάρτησης που ανήκει στο δέντρο. Το εργαλείο αυτό υποστηρίζει τρεις διαφορετικές παραλλαγές σύγκρισης δέντρων κλήσης. Η γενική ιδέα του αλγορίθμου σύγκρισης που αναπτύσσουμε (βλπ Αλγόριθμος 8) βασίζεται στον Αλγόριθμο 6 που παρουσιάστηκε στο Κεφάλαιο 3.



Σχήμα 15: Συγκριτής δέντρων

Ο Αλγόριθμος 6 είναι μια παραλλαγή του αλγόριθμου Levinsthein που εφαρμόζεται σε δέντρα [37]. Πρόκειται για έναν αλγόριθμο σύγκρισης που εκμεταλλεύεται τη γνώση της "γραμματικής". Επειδή η μονάδα σύγκρισης είναι μεμονωμένα σύμβολα, όχι μια ολόκληρη γραμμή, αυτός ο αλγόριθμος μπορεί να επισημάνει τις συντακτικές διαφορές δύο προγραμμάτων. Τα προγράμματα που πρέπει να συγκριθούν με το συγκεκριμένο αλγόριθμο αρχικά μετατρέπονται σε μια παραλλαγή ενός δέντρου ανάλυσης από έναν

αναλυτή. Ένας κόμβος στο δέντρο υποδηλώνει είτε ένα διακριτικό, όπως ένα μεταβλητό όνομα, είτε ένα μη τερματικό που αντιπροσωπεύει μια υποδομή, όπως μια έκφραση.

Η τιμή ομοιότητας των δέντρων, και συνεπώς του πηγαίου κώδικα που αναπαριστούν, ισούται με την τιμή ομοιότητας των δύο ριζών. Το πλεονέκτημα αυτού του αλγορίθμου είναι ότι για τον υπολογισμό της ομοιότητας των ριζών γίνεται σύγκριση σε βάθος. Δηλαδή, μελετάται η ομοιότητα των κόμβων-παιδιών εώς μέχρι το χαμηλότερο επίπεδο του δέντρου.

Η ιδέα του Αλγορίθμου 8 δομείται γύρω από αυτή την προσέγγιση. Σε αυτόν τον αλγόριθμο, η ομοιότητα κάθε κόμβου εξαρτάται από την ομοιότητα των παιδιών του. Οι διαφορές εντοπίζονται στο είδος των δέντρων, στη μέθοδο σύγκρισης των κόμβων και στον τρόπο που αυτά επιλέγονται. Ο αλγόριθμός αυτός χρησιμοποιείται για δέντρα κλήσης συνάρτησης και όχι για δέντρα ανάλυσης. Επίσης, δεν προσμετρούνται στον υπολογισμό της τελικής ομοιότητας μόνο οι πανομοιότυποι κόμβοι αλλά συμπεριλαμβάνεται και η μερική ομοιότητά μεταξύ κόμβων. Τέλος, επειδή μελετάμε δέντρα κλήσης δε λαμβάνουμε υπόψη τη θεώρηση ότι δύο κόμβοι μπορούν να είναι όμοιοι μόνο αν και οι κόμβοι-αδέρφια που βρίσκονται πριν από αυτούς είναι όμοιοι. Αυτό γίνεται διότι η εκτέλεση μιας μεθόδου που έπεται της εκτέλεσης μιας άλλης δεν εξαρτάται υποχρεωτικά από την τελευταία.

Η σύγκριση του πηγαίου κώδικα δύο συναρτήσεων που προτείνουμε χρησιμοποίει την ομοιότητα tanimoto. Το πλεονέκτημα της ομοιότητας τανιμοτο σε σχέση με την ευκλείδεια απόσταση είναι ότι είναι πιο ισχυρή, καθώς κανονικοποιεί το αποτέλεσμα. Η ομοιότητα tanimoto παρήγαγε επίσης καλύτερα αποτελέσματα για το συγκεκριμένο αλγόριθμο σε σχέση με τη συνημιτονική ομοιότητα.

Αλγόριθμος 8 Αλγόριθμος σύγκρισης δέντρων κλήσης

[1.] Λαμβάνονται, αρχικά, δύο δέντρα. Υπολογίζεται η ομοιότητα tanimoto του πηγαίου κώδικα των ριζών σύμφωνα με τη σχέση (2.6).

[2.] Λαμβάνονται τα παιδιά κάθε ρίζας. Ελέγχεται ο αριθμός των παιδιών

[3.] Αν ο αριθμός των παιδιών και των δύο ριζών είναι μηδενικος τότε η ομοιότητα των ριζών ισούται με την ομοιότητα tanimoto του πηγαίου κώδικα των συναρτήσεων. Αν ο αριθμός των παιδιών και της μιας ρίζας είναι μηδενικος ενώ της άλλης μη μηδενικός τότε η ομοιότητα των ριζών ισούται με την ομοιότητα tanimoto του πηγαίου κώδικα των συναρτήσεων διά του αριθμού των παιδιών αυξημένου κατά ένα. Αν ο αριθμός των παιδιών και των δύο ριζών είναι μη μηδενικός τότε κάθε παιδί της μιας ρίζας συγκρίνεται με κάθε παιδί της άλλης ρίζας. Για κάθε σύγκριση των παιδιών καλείται η ίδια διαδικασία.

[4.] Στη συνέχεια, γίνεται η αντιστοίχιση μεταξύ των παιδιών. Αναζητάται το ζευγάρι με τη μεγαλύτερη τιμή ομοιότητας. Αυτό το ζευγάρι αποτελεί ένα ζευγάρι αντιστοίχισης. Οι κόμβοι αυτοί εξαιρούνται από τη νέα αναζήτηση. Η διαδικασία επαναλαμβάνεται εώς ότου όλοι οι κόμβοι-παιδιά τουλάχιστον της μιας ρίζας αντιστοιχηθούν με κόμβους-παιδιά της άλλης ριζας.

[5.] Η ομοιότητα των ριζών ισούται με το άθροισμα της ομοιότητας tanimoto του πηγαίου κώδικα των συναρτήσεων που αντιστοιχούν στις ρίζες και των ομοιοτήτων τον κόμβων-παιδιών βάσει τις αντοιστίχισης διά του αριθμού των περισσότερων παιδιών αυξημένο κατά ένα.

Εφόσον αντιστοιχηθούν οι κόμβοι των δύο δέντρων μεταξύ τους υπολογίζεται η τιμή ομοιότητας μεταξύ των δύο δέντρων. Η τιμή αυτή ισούται με το γινόμενο των τιμών ομοιότητας των αντιστοιχισμένων κόμβων. Τα δέντρα που εξήχθησαν από κάθε έργο λογισμικού θα συγκριθούν με όλα τα δέντρα του άλλου έργου. Κάθε δέντρο του ενός έργου αντιστοιχίζεται με ένα δέντρο του άλλου.

Ο δεύτερος αλγόριθμος που υποστηρίζεται από το εργαλείο είναι ίδιος με το προηγούμενο με μία διαφορά. Η διαφορά βρίσκεται στον τρόπο υπολογισμού ομοιότητας των δύο δέντρων. Η τιμή ομοιότητας των δέντρων ισούται με το μέσο όρο των τιμών ομοιότητας των κόμβων που αντιστοιχήθηκαν αντί του γινομένου.

Ο τρίτος αλγόριθμος διαφέρει σε σχέση με τον πρώτο σε δύο σημεία. Πρόκειται για μια μέθοδο που βασίστηκε και στο [40]. Ο αλγόριθμος της βιβλιογραφίας αυτής υπολογίζει την ομοιότητα δύο αφηρημένων συντακτικών δέντρων. Σε αυτή τη μέθοδο, η ομοιότητα δύο κόμβων όπου δεν είναι προτερματικοί υπολογίζεται από τον τύπο:

$$Similarity(n_1, n_2) = \lambda \prod_{j=1}^{|n_1|} \left(\sigma + similarity(n_{1j}, n_{2j})\right) \quad (4.3α')$$

όπου λ είναι ένας παράγοντας αποσύνθεσης, $|n_1|$ είναι ο αριθμός των παιδιών του n_1 και n_j είναι το j-οστό

παιδί του κόμβου n.

Επομένως στο σύστημά μας η ομοιότητα δύο κόμβων n_1 και n_2 με κόμβους-παιδιά n_{1i} και n_{2j} τα οποία αντιστοιχήθηκαν σε ζεύγη μεταξύ τους δίνεται από τη σχέση:

$$Similarity(n_1, n_2) = \prod_{j=1}^{\min(|n_1|,|n_2|)} \big(tsim(n_1, n_2) + similarity(n_{1j}, n_{2j})\big) \quad (4.36')$$

όπου $|n_1|$, $|n_2|$ είναι το πλήθος των παιδιών των κόμβων n_1 και n_2 αντίστοιχα και n_{1j}, n_{2j} είναι το $j-$οστό ζευγάρι των κόμβων-παιδιών που αντιστοχήθηκαν. Η ομοιότητα δύο δέντρων υπολογίζεται ως το άθροισμα των ομοιοτήτων των κόμβων που αντιστοιχήθηκαν διά της μέγιστης δυνατής ομοιότητας των κόμβων αυτών στην περίπτωση που ήταν πανομοιότυποι.

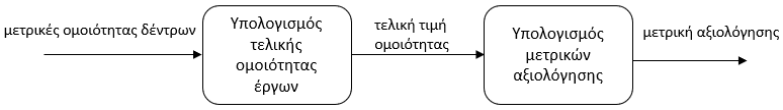
Ο πηγαίος κώδικας κάθε συνάρτησης περιέχει όλα τα απλά σχόλια που βρίσκονται μέσα στον κορμό. Θεωρείται ότι τα σχόλια αυτά χρησιμοποιούνται για να επεξηγήσουν κάτι συγκεκριμένο στον πηγαίο κώδικα, οπότε ο προγραμματιστής είναι δύσκολο να τα αντικαταστήσει. Επίσης, εξετάζεται η συνεισφορά στην ομοιότητα των αντίστοιχων σχολίων τύπου javadoc. Αυτά τα σχόλια παρέχουν μια πιο γενική περιγραφή μιας μεθόδου, οπότε δεν μπορεί να θεωρηθεί με σιγουριά ότι δε θα υπάρξει παραποίηση. Η τιμή ομοιότητας που εξάγεται από τους αλγορίθμους είναι κανονικοποιημένη. Αυτή κυμαίνεται μεταξύ μηδέν και ένα. Η μέγιστη τιμή λαμβάνεται όταν τα υπό σύγκριση δέντρα είναι πανομοιότυπα.

Η σύγκριση των δέντρων υλοποιείται με τις μεθόδους της κλάσης FullComparedTreeAlgorithm. Κατά τη δημιουργία της κλάσης αυτής δίνονται ως ορίσματα δύο αντικείμενα τύπου FileFeatures και δύο ακέραιοι αριθμοί. Οι ακέραιοι αριθμοί δηλώνουν το αναγνωριστικό της ρίζας των δύο δέντρων αντίστοιχα. Τα αρχεία τύπου FileFeatures περιέχουν τις απαραίτητες πληροφορίες των δέντρων. Οι πληροφορίες αυτές είναι οι εξής: Ένας πίνακας κατακερματισμού που δηλώνει για κάθε αναγνωριστικό ενός κόμβου τη λίστα με τα αναγνωριστικά των παιδιών του, ένας πίνακας κατακερματισμού όπου για κάθε αναγνωριστικό είναι αποθηκευμένη μια λίστα με τον πηγαίο κώδικα της συνάρτησης αποθηκευμένο ανά γραμμή και μια λίστα με τα αναγνωριστικά των ριζών των δέντρων που εξήχθησαν από κάθε έργο λογισμικού. Παρόλο που με την αποθήκευση των χαρακτηριστικών όλων των έργων λογισμικού αυξάνεται η χωρική πολυπλοκότητα, μειώνεται σημαντικά η χρονική πολυπλοκότητα της ανάγνωσης και υπολογισμού των χαρακτηριστικών των συγκρινόμενων έργων.

Η υλοποίηση του πρώτου αλγορίθμου πραγματοποιείται με τη μέθοδο compareTrees, ενώ του δεύτερου και του τρίτου με τις μεθόδους compareTrees2 και compareTrees3 αντίστοιχα.

4.2.4 Εξαγωγέας μετρικών

Ο εξαγωγέας μετρικών (βλπ. Σχήμα 16) είναι υπεύθυνος για τον υπολογισμό της τελικής ομοιότητας των έργων λογισμικού, καθώς και για τον υπολογισμό των μετρικών αξιολόγησης του συστήματος.



Σχήμα 16: Εξαγωγέας μετρικών

Για τον υπολογισμό της τελικής ομοιότητας θα πρέπει τα δέντρα για τα οποία έχει πραγματοποιηθεί σύγκριση στο προηγούμενο βήμα να αντιστοιχηθούν μεταξύ τους. Η αντιστοίχιση γίνεται με τρόπο ώστε η τιμή ομοιότητας μεταξύ των δύο δέντρων να είναι μέγιστη. Αυτό συμβαίνει εώς ότου αντιστοιχηθούν όλα τα δέντρα τουλάχιστον του ενός έργου λογισμικού με δέντρα του άλλου. Η τιμή ομοιότητας δύο έργων λογισμικού υπολογίζεται στη συνέχεια ως ο μέσος όρος των τιμών ομοιότητας των αντιστοιχισμένων δέντρων.

Εφόσον δημιουργηθούν οι πίνακες συγκρίνονται όλα τα έργα λογισμικού μεταξύ τους. Στη συνέχεια, αποθηκεύεται το αποτέλεσμα κάθε σύγκρισης και η πληροφορία αν τα έργα λογισμικού που συγκρίνονται περιέχονται στον ίδιο φάκελο, το οποίο δείχνει αν είναι διακλαδώσεις του ίδιου έργου. Μετά την ολοκλήρωση της σύγκρισης εξάγεται μια μετρική αξιολόγησης για αυτό το υποσύστημα.

5 Πειραματικά Αποτελέσματα

Εφόσον πραγματοποιήθηκε η ανάλυση του συστήματός μας, επόμενο βήμα είναι η αξιολόγησή του. Για το σκοπό αυτό διεξάγεται μια σειρά πειραμάτων όπου συγκρίνονται διάφορες μέθοδοι των υποκεφαλαίων 2.1 και 2.3, καθώς κοι οι μέθοδοι δικής μας δημιουργίας που περιγράφηκαν στο προηγούμενο κεφάλαιο.

Για το σκοπό αυτό δημιουργήσαμε ένα σύνολο δεδομένων χρησιμοποιώντας έργα λογισμικού (project) που εξήχθησαν από την αποθήκη λογισμικού GitHub για διάφορα ερωτήματα. Για κάθε έργο λογισμικού που ανακαλύφθηκε δημιουργήθηκε ένας φάκελος που περιλαμβάνει το έργο αυτό και κάποιες διακλαδώσεις (forks) του -ποικίλουν από 4 εώς 11 ανάλογα το έργο- σε αντίστοιχους υποφακέλους. Έτσι, για παράδειγμα, αν διαθέτουμε δύο έργα λογισμικού testA και testB και κάθε έργο διαθέτει 4 διακλαδώσεις η οργάνωση γίνται ως εξής:

dataset

- testA
- testA_master
- testA_fork1
- testA_fork2
- testA_fork3
- testA_fork4
- testB
- testB_master
- testB_fork1
- testB_fork2
- testB_fork3
- testB_fork4

Η έννοια της διακλάδωσης αντιστοιχεί σε μια παραλλαγή του αρχικού έργου. Επομένως, μπορεί να θεωρηθεί ότι κάθε φάκελος περιέχει παρόμοια έργα λογισμικού.

Το σύστημα που περιγράφηκε στο Κεφάλαιο 4, λαμβάνοντας ως είσοδο το μονοπάτι που βρίσκονται οι φάκελοι, θα πρέπει να συγκρίνει όλα τα έργα λογισμικού. Πραγματοποιούνται διάφοροι συνδυασμοί συγκρίσεων ώστε να προκύψει μια εικόνα για το πως αντιδρά ο κάθε αλγόριθμος στη σύγκριση του ίδιου συνόλου δεδομένων. Σε κάθε εκτέλεση του συστήματος, πρώτα συγκρίνονται τα έργα που είναι διακλαδώσεις διαφορετικών έργων λογισμικού -πρακτικά βρίσκονται σε διαφορετικό φάκελο- και στη συνέχεια τα έργα που είναι διακλαδώσεις του ίδιου έργου λογισμικού- βρίσκονται στον ίδιο φάκελο. Για κάθε σύγκριση εξάγεται μια τιμή ομοιότητας μεταξύ του μηδέν και του ένα. Όσο μεγαλύτερη είναι η τιμή ομοιότητας, τόσο πιο όμοια είναι τα έργα. Στη συνέχεια, απαιτείται η αξιολόγηση του συστήματος με κάποια μετρική.

5.1 Αξιολόγηση Συστήματος

Μετρική AUC

Στη στατιστική, η εξέταση Mann-Whitney U είναι μια μη παραμετρική εξέταση της υπόθεσης ότι είναι εξίσου πιθανό μια τυχαία επιλεγμένη τιμή από ένα δείγμα να είναι μικρότερη ή μεγαλύτερη από μια τυχαία επιλεγμένη τιμή από ένα δεύτερο δείγμα. Αυτή η εξέταση μπορεί να χρησιμοποιηθεί για να προσδιοριστεί εάν δύο ανεξάρτητα δείγματα επιλέχθηκαν από πληθυσμούς που είχαν την ίδια κατανομή.

Η εξέταση περιλαμβάνει τον υπολογισμό μιας στατιστικής τιμής, συνήθως ονομάζεται U, της οποίας η κατανομή υπό τη μηδενική υπόθεση (null hypothesis) είναι γνωστή. Στην περίπτωση των μικρών δειγμάτων, η κατανομή είναι σε πίνακα, αλλά για μεγέθη δείγματος πάνω από 20, η προσέγγιση με τη χρήση της κανονικής κατανομής είναι αρκετά καλή.

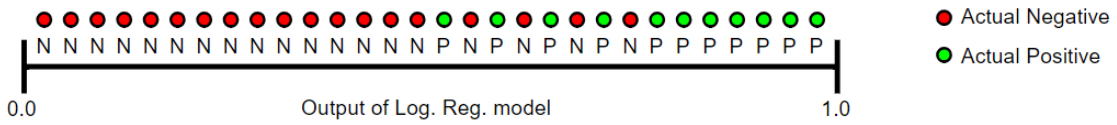
Έτσι, για επαρκώς μεγάλα δείγματα η τιμή U μπορεί να υπολογιστεί με τον ακόλουθο τρόπο. Αρχικά, εκχωρούνται αριθμητικές τάξεις σε όλες τις παρατηρήσεις (τοποθετούνται οι παρατηρήσεις και από τις δύο ομάδες σε μία ομάδα), ξεκινώντας με 1 για τη μικρότερη τιμή. Όπου υπάρχουν ομάδες με την ίδια τιμή, ορίζεται μια τάξη ίση με το μέσο της μη διορθωμένης κατάταξης. Στη συνέχεια, προστίθενται οι τάξεις για τις παρατηρήσεις που προήλθαν από το δείγμα 1. Το άθροισμα των τάξεων στο δείγμα 2 καθορίζεται έπειτα, δεδομένου ότι το άθροισμα όλων των τάξεων είναι N (N + 1) / 2 όπου N είναι ο συνολικός αριθμός

παρατηρήσεων:

$$U_1 = R_1 - \frac{n_1(n_1+1)}{2}$$

όπου n_1 είναι το μέγεθος του δείγματος 1 και R_1 είναι το άθροισμα των τάξεων του ίδιου δείγματος. Αντίστοιχα και για το δεύτερο δείγμα θα ισχύει ότι: $U_2 = R_2 - \frac{n_2(n_2+1)}{2}$. Το άθροισμα αυτών των τιμών ισούται με $U_1 + U_2 = n_1 n_2$.

Με τη βοήθεια των στατιστικών αυτών τιμών υπολογίζεται η μετρική αξιολόγησης AUC. Ένας τρόπος ερμηνείας της μετρικής αυτής είναι η πιθανότητα ότι το μοντέλο κατατάσσει σε υψηλότερη θέση ένα τυχαίο θετικό παράδειγμα περισσότερο από ένα τυχαίο αρνητικό παράδειγμα.



Σχήμα 17: Δείγματα και καθορισμός μετρικής AUC

Η AUC αντιπροσωπεύει την πιθανότητα ένα τυχαίο θετικό (πράσινο) παράδειγμα να είναι τοποθετημένο στα δεξιά ενός τυχαίου αρνητικού (κόκκινο) παράδειγμα (βλπ. Σχήμα17).Η περιοχή τιμών AUC κυμαίνεται από 0 έως 1. Ένα μοντέλο του οποίου οι προβλέψεις είναι 100% λανθασμένες έχει τιμή 0 ενώ αν οι προβλέψεις είναι 100% σωστές, έχει τιμή 1. Η μετρική AUC είναι επιθυμητή για δύο λόγους: α) είναι κλιμακωτά αμετάβλητη, δηλαδή μετράει πόσο καλά ταξινομούνται οι προβλέψεις παρά τις απόλυτες τιμές τους και β) δε μεταβάλλεται από το κατώφλι της ταξινόμησης, οπότε μετράει την ποιότητα των προβλέψεων του μοντέλου ανεξάρτητα από το ποιο είναι το κατώτατο όριο ταξινόμησης.

Βάσει των στατιστικών τιμών U_1 και U_2 η μετρική AUC μπορεί να υπολογιστεί ως:

$$AUC_1 = \frac{U_1}{n_1 n_2} \tag{5.1}$$

Πίνακας Σύγχυσης

Στο πεδίο της μηχανικής μάθησης και ειδικότερα του προβλήματος της στατιστικής ταξινόμησης, ένας πίνακας σύγχυσης, επίσης γνωστός ως πίνακας σφαλμάτων, είναι μια συγκεκριμένη διάταξη πίνακα που επιτρέπει την απεικόνιση της απόδοσης ενός αλγορίθμου. Κάθε σειρά του πίνακα αντιπροσωπεύει τις περιπτώσεις σε μια προβλεπόμενη κλάση ενώ κάθε στήλη αντιπροσωπεύει τις περιπτώσεις σε μια πραγματική κλάση (ή αντίστροφα). Το όνομα πηγάζει από το γεγονός ότι καθιστά εύκολο να δούμε αν το σύστημα προκαλεί σύγχυση σε κάποιες κατηγορίες. Εφόσον οι τιμές ομοιότητας των έργων λογισμικού είναι κανονικοποιημένες, το εύρος τιμών ενός κελιού θα είναι από 0 (τελείως ανόμοια έργα) εώς 1 (πανομοιότυπα έργα).

Χάρτης θερμότητας

Ένας χάρτης θερμότητας (heatmap) είναι μια γραφική παράσταση των δεδομένων όπου οι μεμονωμένες τιμές που περιέχονται σε έναν πίνακα αντιπροσωπεύονται ως χρώματα. Οι χάρτες αυτοί χρησιμοποιούν το χρώμα με τον τρόπο που ένα γράφημα ράβδων χρησιμοποιεί το ύψος και το πλάτος. Έτσι, παρουσιάζουν τα αποτελέσματα ενός πίνακα οπτικοποιημένα ώστε να είναι πιο εύληπτα και κατανοητά. Ο χάρτης θερμότητας εδώ αναπαριστά έναν πίνακα σύγχυσης. Έτσι, όταν ο πίνακας σύγχυσης έχει σε μία θέση την τιμή 0, αυτό αναπαρίσταται με μαύρο χρώμα στην αντίστοιχη θέση στο χάρτη θερμότητας, ενώ όταν έχει την τιμή 1, αναπαρίσταται με λευκό χρώμα.

Κατά την εκτέλεση του συστήματος για ένα σύνολο δεδομένων προκύπτουν δύο δείγματα. Κάθε σύγκριση δύο έργων που είναι διακλαδώσεις διαφορετικών έργων λογισμικού (συμβολίζεται με 0) ανήκει στο ένα δείγμα ενώ κάθε σύγκριση δύο έργων που αποτελούν διακλάδωση του ίδιου έργου λογισμικού (συμβολίζεται με 1) ανήκει στο άλλο δείγμα. Κάθε σύγκριση - πρακτικά κάθε 0 ή 1 - συνοδεύεται από μια τιμή ομοιότητας. Εν συνεχεία, τα δύο δείγματα δημιουργούν ένα ενιαίο δείγμα. Όπως αναφέρθηκε προηγουμένως, θα πρέπει τα στοιχεία του νέου δείγματος να ταξινομηθούν. Η ταξινόμηση πραγματοποιείται

βάσει της τιμής ομοιότητας κατά αύξουσα σειρά. Έτσι, εφόσον αποδοθεί η τάξη σε κάθε στοιχείο του νέου δείγματος μπορούν να υπολογιστούν οι τιμές U_1, U_2 και στη συνέχεια οι μετρικές AUC_1, AUC_2.

Στη συνέχεια, πρέπει να υπολογιστεί ο πίνακας σύγχυσης. Κάθε σειρά λαμβάνει το όνομα ενός φακέλου. Το ίδιο πραγματοποιείται και για κάθε στήλη. Ο πίνακας σύγχυσης στη θέση i, j λαμβάνει μια τιμή ίση με το μέσο όρο των τιμών ομοιότητας των έργων που βρίσκονται στο φάκελο i και συγκρίθηκαν με τις εφαρμογές του φακέλου j. Επομένως, τα στοιχεία της διαγωνίου του πίνακα απεικονίζουν το μέσο όρο των τιμών ομοιότητας που προέκυψαν από τις συγκρίσεις των έργων ενός φακέλου μεταξύ τους. Ο πίνακας αυτός οπτικοποιείται με έναν πίνακα θερμότητας ώστε να γίνονται πιο αντιληπτά τα αποτελέσματα σύγκρισης του κάθε αλγορίθμου.

5.2 Μέθοδοι Συστήματος

Το σύστημά μας υλοποιεί κάποιες μεθόδους της βιβλιογραφίας και κάποιες δικές μας. Αρχικά, μελετάται το περιεχόμενο του πηγαίου κώδικα. Η σύγκριση πηγαίου κώδικα πραγματοποιείται με τις ακόλουθες μεθόδους:

FileTokens [16]. Η μέθοδος αυτή είναι μια μέθοδος της βιβλιογραφίας και ακολουθεί την υλοποίηση που περιγράφεται στο υποκεφάλαιο 2.3 για την εξαγωγή συμβόλων από τα αρχεία των έργων λογισμικού. Για την αφαίρεση των επιθημάτων εφαρμόζεται ο Lovin Stemmer αντί του Porter που περιγράφεται εκεί. Σε κάθε όρο εφαρμόζονται τα βάρη συχνότητας όρου-αντίστροφης συχνότητας αρχείων που περιγράφονται από τη σχέση (4.1) και (4.2). Για τη σύγκριση των διανυσμάτων εφαρμόζεται η συνημιτονική ομοιότητα (2.5).

FunctionTokens. Η μέθοδος αυτή ακολουθεί την ίδια διαδικασία που περιγράφεται παραπάνω με τη διαφορά ότι συγκρίνονται οι συναρτήσεις των έργων λογισμικού, θεωρώντας κάθε συνάρτηση ως ξεχωριστό αρχείο.

LSA [18]. Υπάρχουσα υλοποίηση της λανθάνουσας σημασιολογικής ανάλυσης που ακολουθεί την περιγραφή της παραγράφου 2.1 με τη διαφορά ότι διατηρεί τις λέξεις χωρίς σημασιολογικό περιεχόμενο. Η μέθοδος αυτή συγκρίνει το σημασιολογικό περιεχόμενο των αρχείων του πηγαίου κώδικα. Τη μέθοδο αυτή τη χρησιμοποιούμε ώστε να ελέγξουμε ποιοτικά τον αλγόριθμο FileTokens και για αυτό δεν εφαρμόζεται και σε επίπεδο συναρτήσεων.

LSA+SVD. Η παραπάνω υλοποίηση της λανθάνουσας σημασιολογικής ανάλυσης με ανάλυση σε ιδιάζουσες τιμές. Επιλέχθηκε τάξη $rank = 4$, η οποία βρέθηκε εμπειρικά ότι παράγει τα καλύτερα αποτελέσματα.

CallTreeProd. Εξαγωγή δέντρων κλήσης σύμφωνα με τον Αλγόριθμο 6 και εφαρμογή του Αλγορίθμου 7. Στον πηγαίο κώδικα κάθε συνάρτησης περιέχονται τα απλά σχόλια.

CallTreeSum. Εξαγωγή δέντρων κλήσης και σύγκρισή τους βάσει του ίδιου αλγορίθμου με τη διαφορά ότι η ομοιότητά τους ισούται με το μέσο όρο των τιμών ομοιότητας των κόμβων που αντιστοιχήθηκαν και όχι βάσει του γινομένου αυτών, όπως συνέβαινε προηγουμένως.

CallTreeHeur. Τα δέντρα συγκρίνονται βάσει του ίδιου αλγορίθμου με τη διαφορά ότι η ομοιότητα δύο κόμβων υπολογίζεται με βάση τη σχέση (4.3α'). Επίσης, η ομοιότητα των δέντρων υπολογίζεται ως το άθροισμα των κόμβων που αντιστοιχήθηκαν και κανονικοποιείται με τη μέγιστη δυνατή τιμή που θα μπορούσαν να έχουν στην περίπτωση που η ομοιότητα tanimoto κάθε κόμβου θα ήταν ίση με μονάδα.

Σε κάθε περίπτωση, οι αλγόριθμοι εκτελούνται είτε απευθείας στον πηγαίο κώδικα είτε παίρνοντας υπόψη τα σχόλιά του (+Comments). Για τις μεθόδους FunctionTokens καθώς και για τις μεθόδους που χρησιμοποιούν δέντρα κλήσης συνάρτησης με τον όρο Comments εννοούμε μόνο τα σχόλια τύπου Javadoc. Αυτό συμβαίνει διότι για την πρώτη μέθοδο όλα τα υπόλοιπα σχόλια αφαιρούνται ώστε να μπορεί να γίνει διαχωρισμός των αρχείων σε συναρτήσεις, ενώ για τις μεθόδους που χρησιμοποιούν δέντρα κλήσης συνάρτησης τα σχόλια εντός των συναρτήσεων διατηρούνται σε κάθε περίπτωση καθώς θεωρούνται αρκετά περιγραφικά του κώδικα που συνοδεύουν και η αφαίρεση αυτών για αυτή τη δομή αυξάνει τη χρονική πολυπλοκότητά του.

Οι αλγόριθμοι που προτείνουμε για τη μελέτη της δομής του πηγαίου κώδικα εξελίσσουν ιδέες που υπάρχουν στην υπό μελέτη βιβλιογραφία [16, 37, 40].

5.3 Πειραματικά αποτελέσματα

5.3.1 Σύνολο Δεδομένων υλοποίησης παρόμοιων ζητημάτων

Σε αυτό το υποκεφάλαιο παρουσιάζονται τα πειραματικά αποτελέσματα για το σύνολο δεδομένων που χρησιμοποιήσαμε. Ως σύνολο δεδομένων χρησιμοποιήθηκαν 93 έργα λογισμικού. Τα έργα αυτά προέκυψαν από την αναζήτηση του όρου pacman στη μηχανή αναζήτησης της πλατφόρμας GitHub. Πρόκειται για 10 διαφορετικά έργα λογισμικού repositories και ένα σύνολο από διακλαδώσεις (forks) των έργων αυτών. Οι διακλαδώσεις κάθε έργου λογισμικού ποικίλουν από 4 εώς 11. Τα έργα επιλύουν ζητήματα της ίδιας κατηγορίας ώστε το σύνολο δεδομένων να αποτελεί προσομοίωση μιας πραγματικής κατάστασης και να ελεγχθεί η απόδοση των αλγορίθμων σε πιο απαιτητικές συνθήκες.

Στον Πίνακα 7 παρουσιάζεται η μετρική αξιολόγησης AUC για κάθε αλγόριθμο. Παρατηρούμε ότι η ελάχιστη τιμή της μετρικής αυτής είναι 0.905. Αυτό σημαίνει ότι η διακριτική ικανότητα όλων των αλγορίθμων να διαχωρίσουν τα όμοια από τα ανόμοια έργα λογισμικού είναι υψηλή. Επίσης, για τους αλγορίθμους που δεν εξετάζουν τη δομή του πηγαίου κώδικα χρησιμοποιώντας δέντρα κλήσης συναρτήσεων και λαμβάνουν υπόψη τα σχόλια, η μετρική AUC είναι λίγο πιο υψηλή από ό,τι στους ίδιους αλγορίθμους που δεν τα συμπεριλαμβάνουν στη διαδικασία σύγκρισης. Επομένως, τα σχόλια βοηθούν στο να γίνει καλύτερος διαχωρισμός των όμοιων έργων λογισμικού από τα ανόμοια. Από την άλλη πλευρά, για τις μεθόδους που χρησιμοποιούν δέντρα κλήσης συνάρτησης τα σχόλια τύπου Javadoc δε μεταβάλλουν τη διακριτική ικανότητα τους. Αυτό οφείλεται στο γεγονός ότι η δομή των δέντρων κλήσης συνάρτησης είναι καλώς ορισμένη ώστε να γίνεται αρκετά καλός διαχωρισμός των όμοιων από τα ανομοια έργα λογισμικού χωρίς να επηρεάζεται από την προσθήκη ή όχι των σχολίων τύπου Javadoc.

Η μεγαλύτερη τιμή για τη μετρική αυτή παρατηρείται στην περίπτωση της εφαρμογής του αλγορίθμου της λανθάνουσας σημασιολογικής ανάλυσης καθώς και στην περίπτωση της μεθόδου που συγκρίνει τα αρχεία ως κείμενο και χρησιμοποιεί σύγκριση διανυσμάτων. Οι δύο μέθοδοι στην περίπτωση που τα σχόλια λαμβάνονται υπόψη έχουν την ίδια διακριτική ικανότητα. Στην περίπτωση όμως, που τα σχόλια δε λαμβάνονται υπόψη ο αλγόριθμος της σύγκρισης αρχείων ως κείμενο κάνει ελαφρώς καλύτερο διαχωρισμό από εκείνον του αλγορίθμου της λανθάνουσας σημασιολογικής ανάλυσης. Αυτό σημαίνει, ότι σε αυτή την περίπτωση η στάθμιση συχνότητας όρου-αντίστροφης συχνότητας αρχείων είναι αποδοτικότερη για το διαχωρισμό των όμοιων αρχείων από τα ανόμοια σε σχέση με τη στάθμιση που εφαρμόζει ο αλγόριθμος της λανθάνουσας σημασιολογικής ανάλυσης. Επίσης, η απλή εκδοχή του αλγορίθμου της λανθάνουσας σημασιολογικής ανάλυσης δίνει καλύτερα αποτελέσματα από την περίπτωση που εφαρμόζεται και ο αλγόριθμος ανάλυσης σε ιδιάζουσες τιμές. Αυτό, συμβαίνει διότι κατά τη μείωση των διαστάσεων χάνεται και χρήσιμη πληροφορία.

Επίσης, η μέθοδος CallTreeProd έχει μεγαλύτερη διακριτική ικανότητα σε σχέση με την CallTreeSum. Αυτό μπορεί να εξηγηθεί από το γεγονός ότι στον πρώτο αλγόριθμο κάθε διαφορά μεταξύ δύο κόμβων έχει μεγαλύτερο αντίκτυπο στο τελικό αποτέλεσμα, καθώς βασίζεται σε γινόμενο αντί για άθροισμα ομοιοτήτων. Αυτό, οδηγεί σε σχεδόν μηδενική ομοιότητα σε έργα λογισμικού που δεν έχουν σχέση μεταξύ τους και άρα καλύτερο διαχωρισμό.

Ακόμη, η μέθοδος FileTokens πραγματοποιεί καλύτερο διαχωρισμό σε σχέση με τη FunctionTokens στην περίπτωση που τα σχόλια λαμβάνονται υπόψη. Αυτό, συμβαίνει διότι στην πρώτη μέθοδο η αντιστοίχιση μεταξύ των αρχείων είναι αποδοτικότερη από ό,τι η αντιστοίχιση μεταξύ των συναρτήσεων στη δεύτερη. Συγκεκριμένα, η πρώτη μέθοδος λαμβάνει υπόψη τις δηλώσεις εισαγωγής βιβλιοθηκών καθώς και τα σχόλια που βρίσκονται εντός του αρχείου και εκτός των συναρτήσεων, όπως για παράδειγμα μια συνάρτηση που για να εξαιρεθεί από τη διαδικασία εκτέλεσης περικλείεται από τα σύμβολα των σχολίων. Επίσης, η μέθοδος FunctionTokens βρίσκει τις ομοιότητες μεταξύ των συναρτήσεων ανεξάρτητα από το αρχείο προέλευσής τους. Συνεπώς, ανόμοια μεταξύ τους έργα λογισμικού μπορούν να χρησιμοποιούν κάποιες συναρτήσεις κοινές, όπως μια συνάρτηση για την εύρεση μέγιστης τιμής, με αποτέλεσμα να δίνεται μεγαλύτερη ομοιότητα σε τέτοια έργα και ο διαχωρισμός να είναι ελαφρώς καλύτερος για τη μέθοδο FileTokens. Στην περίπτωση που αφαιρεθούν τα σχόλια, η μέθοδος FunctionTokens είναι εκείνη που κάνει ελαφρώς καλύτερο διαχωρισμό σε σχέση με τη μέθοδο FileTokens. Με την αφαίρεση κάθε είδους σχολίων χάνεται πολύτιμη πληροφορία και επιδρά αρκετά αρνητικά στη διακριτική ικανότητα της μεθόδου FileTokens. Από την άλλη, στη μέθοδο FunctionTokens υπάρχει η απώλεια μόνο των σχολίων τύπου Javadoc- καθώς τα υπόλοιπα σχόλια αφαιρούνται κατά τη διαδικασία διάσπασης των αρχείων σε συναρτήσεις- με αποτέλεσμα η χρήσιμη πληροφορία που χάνεται να είναι λιγότερη και η επίδραση της

απώλειας των σχολίων στη διακριτική ικανότητα αυτής της μεθόδου να είναι μικρότερη. Έτσι, ο υπολογισμός της ομοιότητας μεταξύ των αρχείων στην περίπτωση της μεθόδου FileTokens στηρίζεται μόνο στις συναρτήσεις και στις δηλώσεις εισαγωγής βιβλιοθηκών που διαθέτει. Επομένως, εφόσον το κυριάρχο κριτήριο και στις δύο μεθόδους για τον υπολογισμό της ομοιότητας δύο έργων είναι ο πηγαίος κώδικας των συναρτήσεων, η μέθοδος FunctionTokens υπολογίζει με μεγαλύτερη ακρίβεια της όμοιες συναρτήσεις.

Σε κάθε περίπτωση, η ύπαρξη της μεθόδου FunctionTokens κρίνεται απαραίτητη καθώς αντιλαμβάνεται με μεγαλύτερη επιτυχία τις περιπτώσεις όπου υπάρξει ανακατανομή στις συναρτήσεις. Για παράδειγμα, έστω ότι για δύο έργα λογισμικού με δύο αρχεία το καθένα η FileTokens δίνει πλήρη ομοιότητα. Αν γίνει μια ανακατανομή στις συναρτήσεις που υπάρχουν στα δύο αρχεία στο ένα από τα δύο έργα και εφαρμοστεί η ίδια διαδικασία σύγκρισης, η τιμή ομοιότητας θα μειωθεί. Αν όμως εφαρμοστεί η μέθοδος FunctionTokens, η τιμή ομοιότητας θα είναι η ίδια και στις δύο περιπτώσεις.

Μέθοδος	AUC με σχόλια	AUC χωρίς σχόλια
FileTokens	0.991	0.985
FunctionTokens	0.988	0.986
LSA	0.991	0.977
LSA+SVD	0.970	Μη σύγκλιση
CallTreeProd	0.972	0.972
CallTreeSum	0.906	0.905
CallTreeHeu	0.929	0.929

Πίνακας 7: Αξιλόγιση αλγορίθμων για σύγκριση παρόμοιων συστημάτων

Στα σχήματα 18 εώς 24 παρουσιάζονται οι χάρτες θερμότητας για κάθε μέθοδο. Οι χάρτες αυτοί αποτελούν μια οπτικοποίηση των πινάκων σύγχυσης για κάθε μέθοδο που βρίσκεται στο παράρτημα Α. Στην καλύτερη περίπτωση, οι πίνακες σύγχυσης πρέπει να είναι μοναδιαίοι, όπου οι αντίστοιχοι χάρτες θερμότητας έχουν λευκό χρώμα μόνο στην κύρια διαγώνιο όσο περισσότερο μαύρο χρώμα αλλού.

Αρχικά, παρατηρείται από τους πίνακες σύγχυσης ότι κατά κύριο λόγο η ομοιότητα των έργων λογισμικού που αποτελούν διακλαδώσεις του ίδιου έργου είναι μεγαλύτερη στις μεθόδους που αξιοποιούν τα σχόλια από ό,τι στις ίδιες μεθόδους όταν τα αφαιρούν. Η ομοιότητα όμως των έργων που είναι διακλαδώσεις διαφορετικών αρχείων είναι μικρότερη στις μεθόδους που χρησιμοποιούν τα σχόλια από ό,τι στις ίδιες μεθόδους όταν δεν τα χρησιμοποιούν. Επομένως, τα σχόλια επηρεάζουν θετικά την ομοιότητα των αρχείων.

Στο Σχήμα18 βρίσκονται οι χάρτες θερμότητας για τη μέθοδο σύγκρισης αρχείων ως κείμενα. Παρόλο που η κύρια διαγώνιος έχει λευκό χρώμα (οι αντίστοιχοι πίνακες έχουν τιμές αρκετά κοντά στη μονάδα στις περισσότερες θέσεις της κύριας διαγωνίου), παρατηρείται ότι οι υπόλοιπες θέσεις του χάρτη δεν έχουν τόσο σκούρο χρώμα σε σχέση με άλλες μεθόδους. Συγκεκριμένα, στον πίνακα σύγχυσης 9 του αλγορίθμου FileTokens, η μέγιστη μέση τιμή σύγκρισης έργων λογισμικού είναι 0.327, ενώ επίσης για όλες τις συγκρίσεις έργων που είναι διασυνδέσεις διαφορετικών έργων λογισμικού οι τιμές ομοιότητας δεν κινούνται σε μηδενικά επίπεδα. Αυτό δικαιολογείται στο συγκεκριμένο αλγόριθμο καθώς ο ίδιος λαμβάνει υπόψη μόνο το περιεχόμενο αναπαριστώντας τον πηγαίο κώδικα ως σακούλες λέξεων. Όλα τα έργα λογισμικού υλοποιούν όχι απλά όμοια ζητήματα, αλλά συγκεκριμένο παιχνίδι, το pacman. Αυτό σημαίνει ότι υπάρχουν λέξεις-κλειδιά που προκύπτει από την ορολογία του παιχνιδιού, όπως player, ghost, board που έχουν χρησιμοποιηθεί από όλα σχεδόν τα έργα. Έτσι, το μόνο που διαφέρει είναι οι συχνότητες εμφάνισης των όρων. Για αυτό, παρατηρούνται σε ανόμοια αρχεία μη μηδενικές τιμές, χωρίς ωστόσο να είναι επαρκώς μεγάλες.

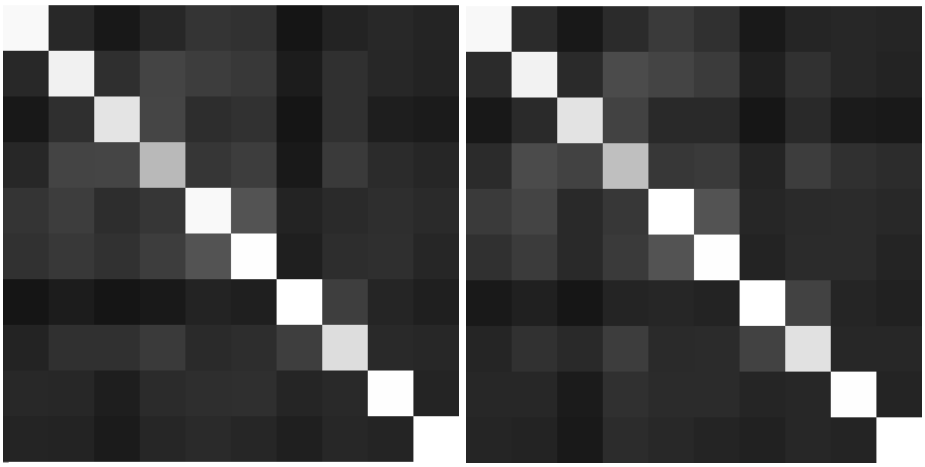
Στον Πίνακα 11, όπου γίνεται η ίδια διαδικασία σε επίπεδο συνάρτησης παρατηρούνται εξίσου υψηλές τιμές ομοιότητας που αγγίζουν τη μονάδα στη διαγώνιό του, εφόσον η αντιστοίχιση γίνεται με μεγαλύτερη ακρίβεια. Συγκεκριμένα, οι ομοιότητες στην κύρια διαγώνιο για αυτήν τη μέθοδο είναι πιο υψηλές από ό,τι για τη μέθοδο fileTokens. Αυτό σημαίνει ότι υπήρξαν, ενδεχομένως, ανακατατάξεις των συναρτήσεων στα αρχεία. Εκτός, από τη μεγαλύτερη ομοιότητα στα έργα λογισμικού της κύριας διαγωνίου που είναι μεταξύ τους όμοια, παρατηρείται αυξημένη ομοιότητα και στα ανόμοια αρχεία. Όπως, ειπώθηκε και πριν, διαφορετικά έργα λογισμικού έχουν σε διαφορετικά αρχεία συναρτήσεις που υλοποιούν όμοιες λειτουργίες, όπως την εύρεση μια μέγιστης τιμής ή τον υπολογισμό μέσου όρου ενός πίνακα. Επειδή, όλα τα έργα λογισμικού υλοποιούν όμοια προβλήματα είναι περισσότερες και οι παρόμοιες συναρτήσεις

που χρησιμοποιούνται, όπως η επιστροφή του αναγνωριστικού του παίκτη ή του σκορ του παιχνιδιού. Επομένως, η συγκεκριμένη μέθοδος, δρα συμπληρωματικά με τη μέθοδο fileTokens ή τη μέθοδο της λανθάνουσας σημασιολογικής ανάλυσης ώστε να προκύψει μια ξεκάθαρη εικόνα σχετικά με την ομοιότητα των έργων.

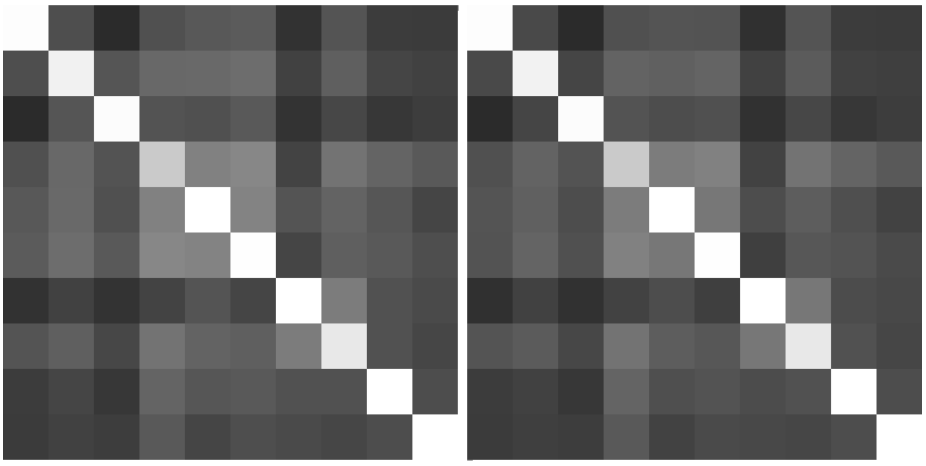
Όπως φαίνεται από το Σχήμα20, ο απλός αλγόριθμος λανθάνουσας σημασιολογικής ανάλυσης δίνει επαρκώς μικρές τιμές στα έργα διαφορετικών φακέλων. Συγκεκριμένα, καμία τιμή ομοιότητας δύο έργων που βρίσκονται σε διαφορετικό φάκελο δεν ξεπερνά το 0.118. Οι τιμές ομοιότητας της κυρίας διαγωνίου είναι επαρκώς μεγάλες αλλά μικρότερες σε σχέση με τις τιμές που δίνει ο αλγόριθμος fileTokens. Αν όμως εφαρμοστεί ο αλγόριθμος ανάλυσης του πίνακα σε ιδιάζουσες τιμές, παρόλο που η διακριτική ικανότητά του είναι υψηλή, οι τιμές ομοιότητας που δίνει ακόμη και σε πανομοιότυπα έργα λογισμικού είναι αρκετά μικρές και μη αντιπροσωπευτικές. Το γεγονός αυτό είναι αναμενόμενο, εφόσον πραγματοποιήθηκε μείωση διαστάσεων και συνεπώς αλλοίωση της πληροφορίας για τη διεξαγωγή αποτελεσμάτων.

Τα σχήματα 22 εώς 24 αποτελούν τους χάρτες θερμότητας για τις τρεις παραλλαγές του αλγορίθμου δέντρων κλήσης. Παρατηρείται ότι οι αλγόριθμοι αυτή δίνουν επαρκώς μεγάλες τιμές ομοιότητας σε έργα λογισμικού που είναι διακλαδώσεις του ίδιου έργου και σχεδόν μηδαμινές σε έργα που είναι διακλαδώσεις διαφορετικών έργων. Αυτό σημαίνει ότι η δομή του πηγαίου κώδικα έτσι όπως προκύπτει από τα δέντρα κλήσης είναι καταλυτικός παράγοντας για την ομοιότητα δύο έργων λογισμικού. Αν μελετήσουμε και τους αντίστοιχους πίνακες σύγχυσης παρατηρούμε ότι η μέθοδος CallTreeProd+Comments για την οποία έχουμε και τη μεγαλύτερη τιμή AUC για τις τρεις αυτές παραλλαγές, θέτει την τιμή μηδέν σε πολλά ανόμοια μεταξύ τους έργα λογισμικού. Επειδή, η μέθοδος αυτή στηρίζεται στο γινόμενο των τιμών ομοιότητας των αντιστοιχισμένων κόμβων, επηρεάζεται μόνο από τις διαφορές που υπάρχουν μεταξύ των δέντρων κλήσης. Για παράδειγμα, έστω ότι έχουμε δύο πολύ απλά δέντρα κλήσης που αποτελούνται συνολικά από δύο κόμβους: τη ρίζα και το παιδί της. Αν ο κώδικας των συναρτήσεων που αντιστοιχούν στα παιδιά είναι πανομοιότυπος τότε οι κόμβοι αυτοί έχουν ομοιότητα ίση με τη μονάδα. Αν οι κώδικες των συναρτήσεων δεν έχουν την παραμικρή ομοιότητα τότε η ομοιότητα των κόμβων θα έτσι όπως ορίστηκε από τον Αλγόριθμο 6, θα ισούται με 0.5. Έτσι, η τελική ομοιότητα των δέντρων θα ισούται με 0.5. Βλέπουμε, δηλαδή, ότι οι πανοποιότυποι κόμβοι δεν επηρεάζουν το αποτέλεσμα, απλά το διατηρούν.

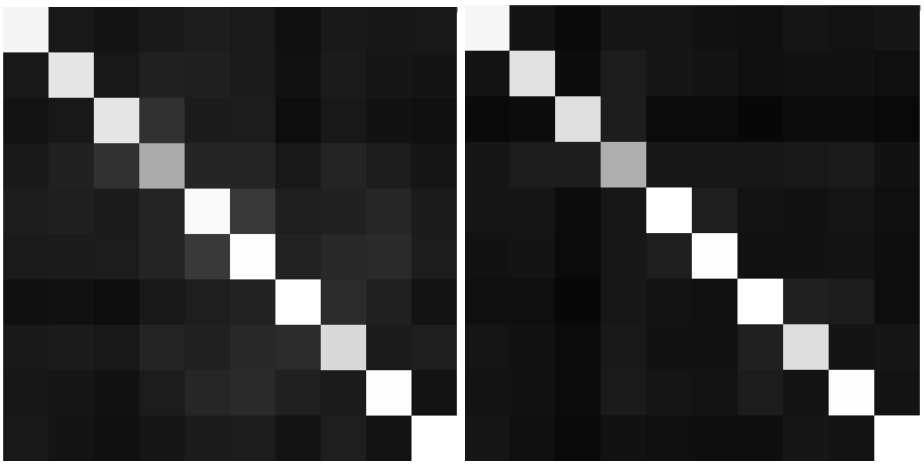
Ενδιαφέρον προξενεί το γεγονός ότι έργα λογισμικού που αποτελούν διακλαδώσεις του ίδιου έργου έχουν τιμές ομοιότητας με μεγάλη απόκλιση σε διαφορετικούς αλγορίθμους. Για παράδειγμα, τα έργα λογισμικού του φακέλου testB στον Πίνακα 13 του αλγορίθμου λανθάνουσας σημασιολογικής ανάλυσης έχουν τιμή ομοιότητα 0.884 ενώ στον Πίνακα 16 του αλγορίθμου κλήσης δέντρων CallTreeProd+Comments 0.389. Σε αυτόν το φάκελο υπάρχουν έργα λογισμικού με εντελώς διαφορετικό αριθμό συναρτήσεων. Συγκρίνονται, δηλαδή, έργα που διαθέτουν 90 συναρτήσεις με άλλα που διαθέτουν 205 συναρτήσεις. Αυτά τα δύο έργα όπως είναι φυσικό, δεν έχουν τα ίδια δέντρα κλήσης συναρτήσεων. Επομένως, οδηγούν και σε μικρή τιμή ομοιότητας. Ο αλγόριθμος, όμως, της λανθάνουσας σημασιολογικής ανάλυσης λαμβάνει υπόψη τις όμοιότητες των αντιστοιχισμένων αρχείων, οπότε η ομοιότητα είναι δικαιολογημένα πιο αυξημένη. Καλούνται, δηλαδή, συναρτήσεις που βρίσκονται σε νέα αρχεία τα οποία δεν αντιστοιχήθηκαν με κάποιο αρχείο ενός άλλου έργου. Έτσι τα δέντρα κλήσης και η δομή του έργου λογισμικού να διαφέρουν από έργο σε έργο.



Σχήμα 18: Πίνακες θερμότητας FileTokens (αριστερά) και FileTokens+Comments (δεξιά)



Σχήμα 19: Πίνακες θερμότητας FunctionTokens (αριστερά) και FunctionTokens+Comments (δεξιά)



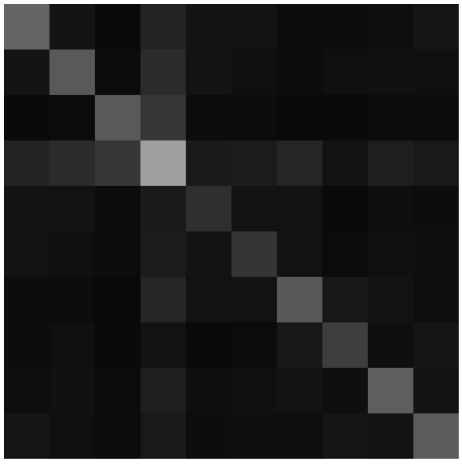
Σχήμα 20: Πίνακες θερμότητας LSA (αριστερά) και LSA+Comments (δεξιά)

5.3.2 Σύνολο δεδομένων υλοποίησης διαφορετικών ζητημάτων

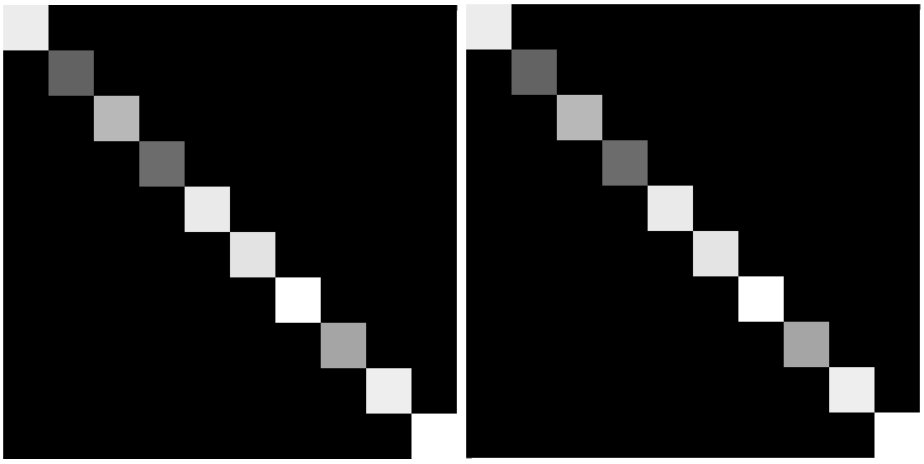
Στη συνέχεια, εφαρμόστηκε ένα νέο σύνολο δεδομένων αποτελούμενο απο 4 φακέλους. Η οργάνωση του συνόλου είναι ίδια με αυτή που περιγράφηκε παραπάνω. Κάθε φάκελος περιέχει έργα λογισμικού που υλοποιούν το ίδιο ζήτημα και επρόκειτο για ένα project της πλατφόρμας GitHub με κάποια forks.Το σύνολο δεδομένων αποτελείται από τους εξής φακέλους: έναν φάκελο έργων που αφορα τη στοίβα διεπαφής για android, έναν φάκελο σχετικά με λίστες διεπαφής επίσης για android,έναν φάκελο όπου τα έργα σχετίζονται με την ανάλυση παλινδρόμησης (logistic regression) και έναν φάκελο από το προηγούμενο σύνολο δεδομένων που αφορά το pacman. Σε αυτό το σύνολο δεδομένων εφαρμόζονται 4 από τους αλγόριθμους που υλοποιήσαμε και λαμβάνουν υπόψη τα σχόλια ώστε να εξετάσουμε τη διακριτική τους ικανότητα και στην περίπτωση που οι φάκελοι με τα έργα λογισμικού διαφέρουν αρκετά μεταξύ τους. Οι αλγόριθμοι αυτοί επιλέχθηκαν με βάσει τη μετρική AUC που είχαν για το προηγούμενο σύνολο δεδομένων.

Ο πίνακας 8 δείχνει τη μετρική AUC για το σύνολο δεδομένων. Παρατηρείται ότι για όλες τις μεθόδους η μετρική είναι ίση με τη μονάδα. Αυτό σημαίνει ότι γίνεται τέλειος διαχωρισμός μεταξύ των έργων λογισμικού παρόλο που, όπως θα δούμε παρακάτω στους χάρτες θερμότητας, το κάθε έργο με τα αντίστοιχα forks του δεν είναι πανομοιότυπα. Το αποτέλεσμα αυτό είναι λογικό εφόσον τα έργα που αποτελούν διασυνδέσεις διαφορετικών έργων υλοποιούν διαφορετικά ζητήματα και επομένως δεν υπάρχει ομοιότητα ούτε σε επίπεδο περιεχομένου (χρησιμοποιούνται διαφορετικοί όροι) αλλά ούτε σε επίπεδο δομής (τα δέντρα κλήσης είναι τελείως διαφορετικά μεταξύ τους).

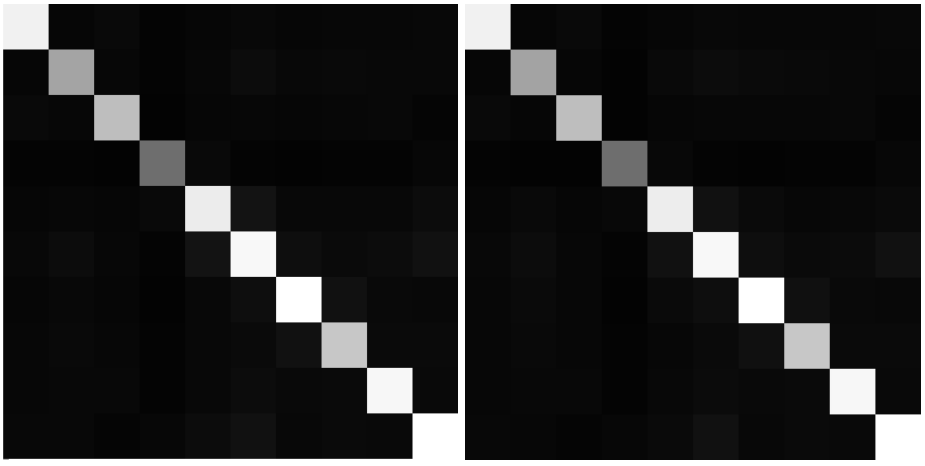
Στο Σχήμα 25 παρουσιάζονται οι χάρτες θερμότητας για τις παραπάνω μεθόδους. Για τη μέθοδο σύγκρισης αρχείων ως κείμενο παρατηρούμε ότι τα έργα λογισμικού που αποτελούν διακλάδωση του ίδιου έργου έχουν μεγάλη ομοιότητα ενώ οι διακλαδώσεις διαφορετικών έργων μικρή. Η ομοιότητα των



Σχήμα 21: Πίνακες θερμότητας LSA+SVD+Comments



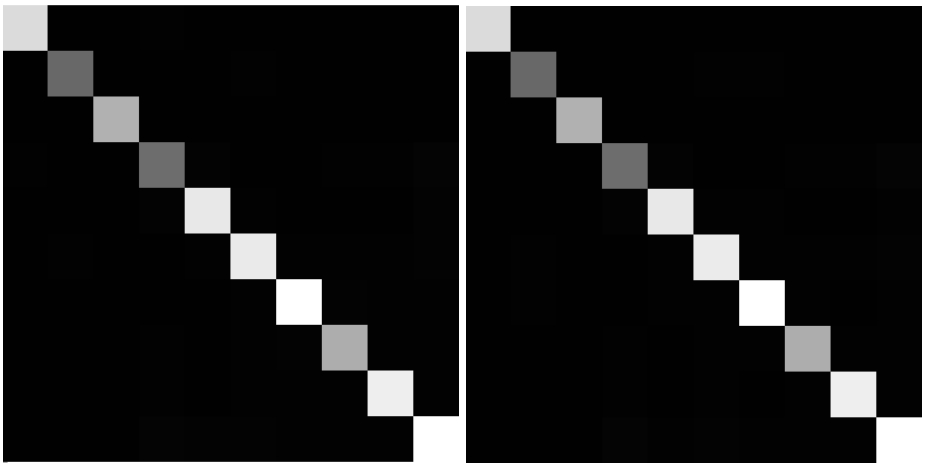
Σχήμα 22: Πίνακες θερμότητας CallTreeProd (αριστερά) και CallTreeProd+Comments (δεξιά)



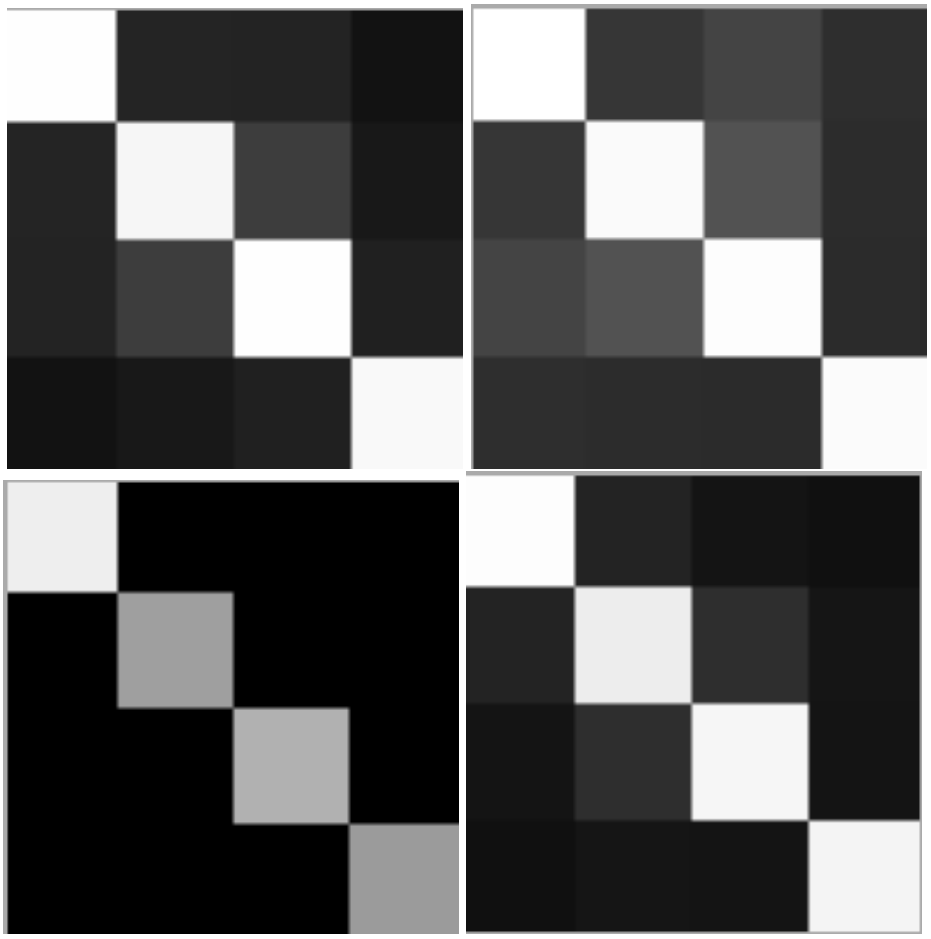
Σχήμα 23: Πίνακες θερμότητας CallTreeSum (αριστερά) και CallTreeSum+Comments (δεξιά)

Μέθοδος	Μετρική AUC
FileTokens+Comments	1.0
FunctionTokens+Comments	1.0
CallTreeProd+Comments	1.0
LSA+Comments	1.0

Πίνακας 8: Αξιολόγιση αλγορίθμων για σύγκριση διαφορετικών συστημάτων



Σχήμα 24: Πίνακες θερμότητας CallTreeHeur (αριστερά) CallTreeHeur+Comments (δεξιά)



Σχήμα 25: Πίνακες θερμότητας FileTokens+Comments (πάνω αριστερά), FunctionTokens+Comments (πάνω δεξιά), CallTreeProd+Comments (κάτω αριστερά) και LSA+Comments (κάτω δεξιά) για σύγκριση διαφορετικών συστημάτων

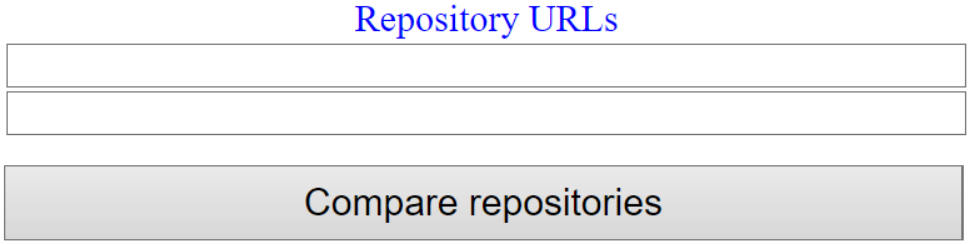
διακλαδώσεων διαφορετικών έργων, κυμαίνεται σε χαμηλότερα επίπεδα σε σχέση με τις διακλαδώσεις διαφορετικών έργων του προηγούμενου συνόλου δεδομένων. Αυτό εξηγείται από τα διαφορετικής φύσεως προβλήματα που υλοποιούν τα έργα. Συνεπώς, δε χρησιμοποιούνται παρόμοιοι όροι. Η μέθοδος σύγκρισης συναρτήσεων ως κείμενο αποδίδει κατά κύριο λόγο μεγαλύτερες τιμές ομοιότητας στις συγκρίσεις όλων των έργων λογισιμικού σε σχέση με την ίδια μέθοδο όταν εφαρμόζεται σε επίπεδο αρχείου για τους ίδιους λόγους που συμβαίνει και στο προηγούμενο σύνολο δεδομένων.

Η μέθοδος CallTreeProd+Comments δίνει μηδενική ομοιότητα στις διακλαδώσεις διαφορετικών έργων λογισμικού, ενώ στις διακλαδώσεις του ίδιου έργου σχετικά υψηλή ομοιότητα. Βλέπουμε ότι η ομοιότητα των διακλαδώσεων του ίδιου έργου κυμαίνεται σε αυτό το σύνολο δεδομένων μεταξύ 0.607 και 0.932. Ο λόγος που συμβαίνει αυτό είναι ότι τα έργα που επιλέχθηκαν έχουν αρκετές διαφορές στη δομή ώστε να ελεγχθεί η διακριτική ικανότητα του αλγορίθμου για αυτήν την περίπτωση.

6 Υπηρεσία Ιστού Υπολογισμού Ομοιότητας

Για την παροχή του συστήματος που αναπτύχθηκε σε τελικούς χρήστες, χρησιμοποιήθηκε μια υπάρχουσα πλατφόρμα[3], η οποία επιτρέπει την εύκολη διανομή προγραμμάτων Java ως RESTful υπηρεσίες. Στη συνέχεια αναπτύχθηκε γραφική διεπαφή, η οποία μπορεί να καλέσει την υπηρεσία αυτή. Στο τελικό πρόγραμμα που προβάλλεται από την RESTful υπηρεσία ενσωματώθηκαν οι αλγόριθμοι FileTokens, FunctionTokens, LSA και CallTreeProd. Και για τους τέσσερις αλγορίθμους λαμβάνονται υπόψη τα σχόλια, βάσει των αποτελεσμάτων του Πίνακα 7.

Η γραφική διεπαφή παρουσιάζεται στο Σχήμα26. Η συγκεκριμένη υπηρεσία δέχεται ως είσοδο την ηλεκτρονική διεύθυνση (url) δύο έργων λογισμικού της πλατφόρμας GitHub. Όταν ο χρήστης πατήσει το κουμπί *Compare Repositories* τα έργα αυτά αποθηκεύονται σε τοπικούς φακέλους. Στη συνέχεια, πραγματοποιείται η σύγκριση των έργων λογισμικού σύμφωνα με τους αλγορίθμους που αναφέρθηκαν παραπάνω. Μετά το πέρας των συγκρίσεων εξάγονται γραφήματα ράβδων. Κάθε γράφημα υποδηλώνει την τιμή ομοιότητας που αποδίδει ο αντίστοιχος αλγόριθμος κατά τη σύγκριση των δύο έργων. Έτσι, εξάγονται συμπεράσματα σχετικά με τους άξονες ομοιότητάς τους. Παρακάτω ακολουθούν παραδείγματα για κάποια έργα λογισμικού και τα συμπεράσματα που μπορούμε να εξάγουμε με βάση τα αποτελέσματα του συστήματός μας.



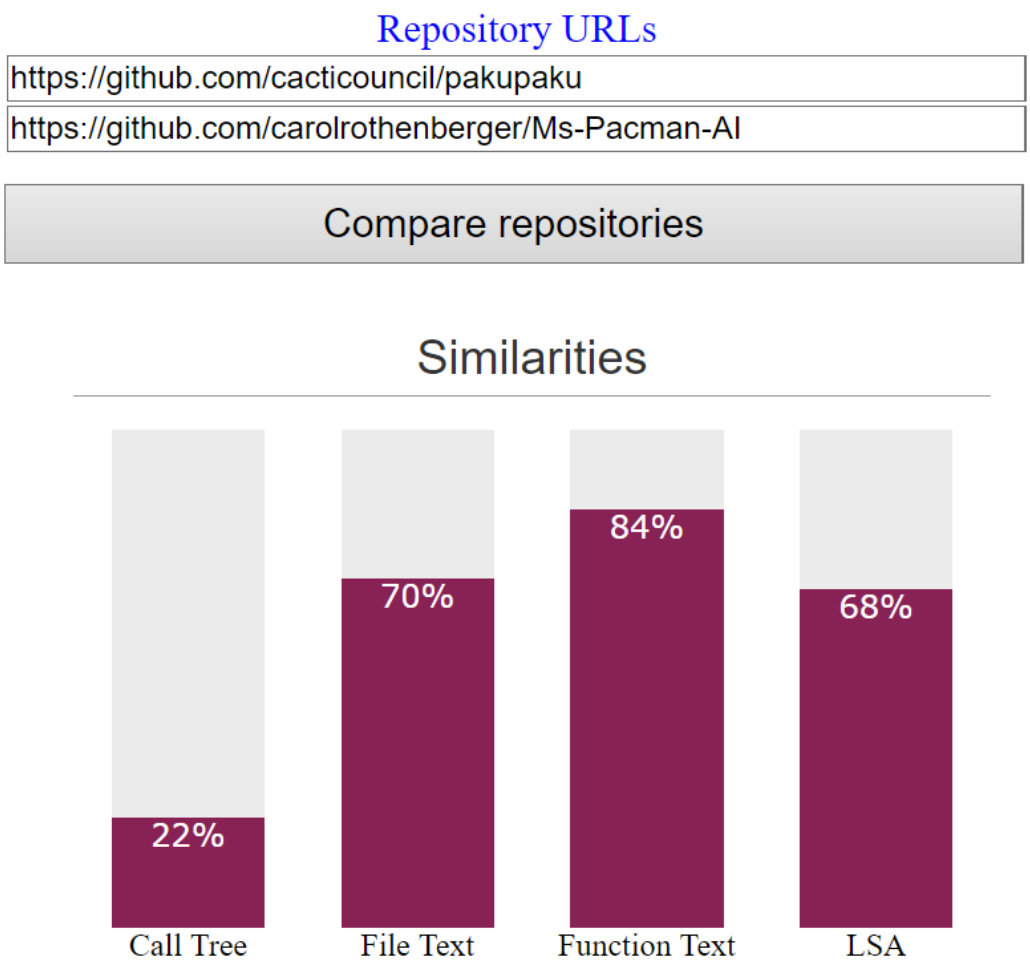
Σχήμα 26: Γραφική διεπαφή του συστήματος

Στο Σχήμα 27 συγκρίνουμε δύο έργα λογισμικού τα οποία αποτελούν διακλαδώσεις του ίδιου έργου. Τα έργα αυτά συγκρίνονται με βάση τις μεθόδους υπολογισμού ομοιότητας CallTreeProd+Comments (Call Tree), FileTokens+Comments (File Text), FunctionTokens+Comments (Function Text) και LSA+Comments (LSA) και το σύστημά μας εμφάνισε αντίστοιχα ραβδογράμματα. Παρατηρούμε ότι η μέθοδος FileTokens υπολογίζει ομοιότητα 70%. Αυτό σημαίνει ότι τα αρχεία που βρίσκονται στα δύο έργα λογισμικού έχουν τροποποιηθεί μερικώς, είτε με την προσθήκη νέων συναρτήσεων είτε με την τροποποίηση των υπαρχόντων. Εφόσον και η μέθοδος LSA δίνει παρόμοιο ποσοστό ομοιότητας, επιβεβαιώνεται η αξιοπιστία της προηγούμενης μεθόδου και οι διαφορές που υπάρχουν μεταξύ των έργων. Η μέθοδος FunctionTokens δίνει ομοιότητα 84%, το οποίο σημαίνει ότι ένα μεγάλο μέρος των συναρτήσεων και στα δύο έργα είναι όμοιο ενώ ενα μικρότερο ποσοστό έχει τροποποιηθεί. Τέλος, το ποσοστό ομοιότητας που αποδίδει η μέθοδος μελέτης δομής του κώδικα CallTreeProd είναι 22%. Αυτό συνεπάγεται ότι η δομή του έργου έχει μεταβληθεί αρκετά. Καθώς η τιμή αυτή διαφέρει πολύ από την τιμή του αλγορίθμου FunctionTokens, μπορούμε να καταλάβουμε ότι έχει γίνει σημαντική αναδιάρθρωση του κώδικα με προσθήκη και αφαίρεση κλήσεων συναρτήσεων, το οποίο έχει επιδράσει στη δομή του γράφου κλήσης, Συνεπώς, μπορούμε να συμπεράνουμε ότι πρόκειται για δύο έργα όπου η βάση τους είναι όμοια αλλά έχουν διαφορετική λογική.

Τα παραπάνω συμπεράσματα επιβεβαιώνονται από το περιεχόμενο των αρχείων των έργων αυτών. Το πρώτο έργο αποτελείται από 48 αρχεία ενώ το δεύτερο από 57 αρχεία. Επίσης, ενώ η κύρια ροή του πρώτου έργου αναπαρίσταται με 10 δέντρα κλήσης συνάρτησης, η κύρια ροή του δεύτερου έργου αναπαρίσταται με 15 δέντρα κλήσης συνάρτησης. Από τα 10 δέντρα κλήσης συνάρτησης του πρώτου έργου λογισμικού, οι ρίζες εφτά δέντρων έχουν παρόμοια ρίζα στα δέντρα του δεύτερου έργου λογισμικού, ενώ οι ρίζες τεσσάρων δέντρων από αυτά έχουν διαφορετικό αριθμό παιδιών.

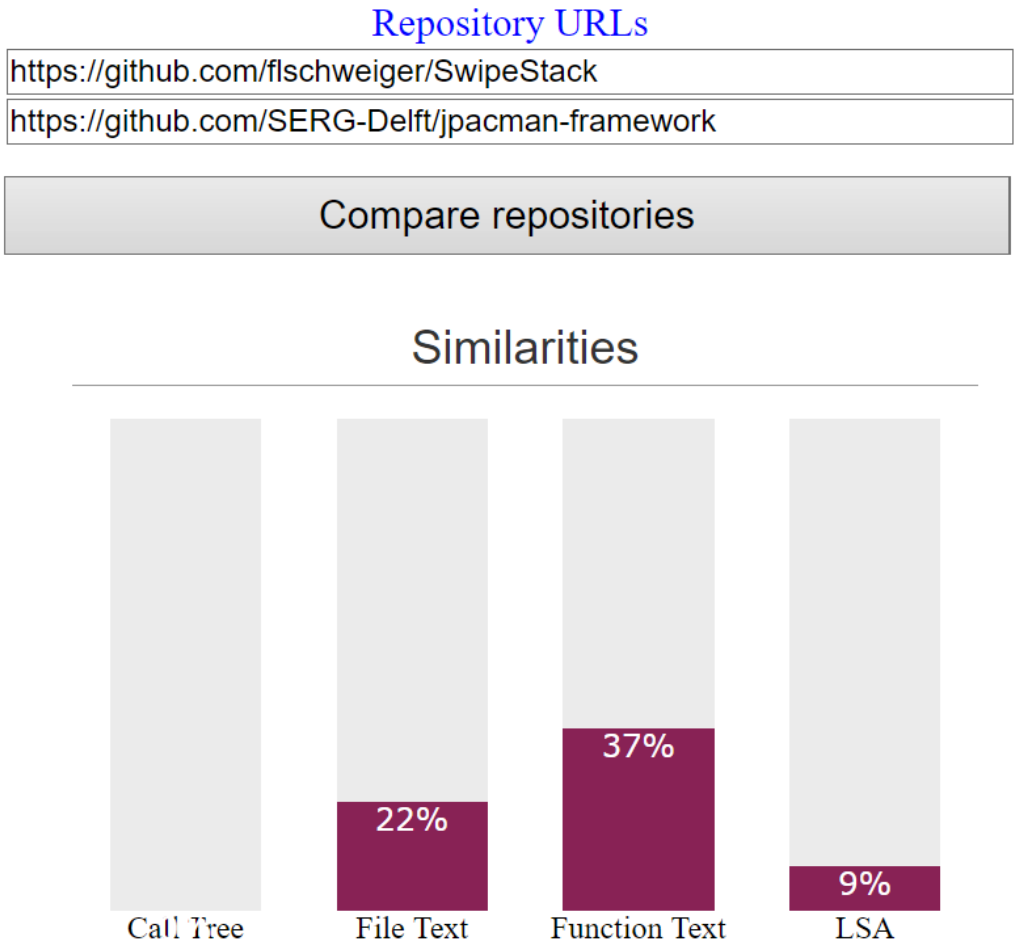
Στη συνέχεια, στο Σχήμα 28 εκτελείται η σύγκριση δύο ασυσχέτιστων έργων λογισμικού. Μπορούμε να δούμε ότι η μέθοδος FileTokens δίνει ποσοστό ομοιότητας 23%. Η μέθοδος LSA αποδίδει ακόμα μικρότερη ομοιότητα για τα δύο έργα λογισμικού και συγκεκριμένα 9%, ενώ η μέθοδος FunctionTokens αποδίδει

[3] https://github.com/maniospas/JAppService



Σχήμα 27: Αποτελέσματα σύγκρισης παρόμοιων έργων

μεγαλύτερη ομοιότητα 35%. Από τις μεθόδους αυτές συμπεραίνουμε ότι τα αρχεία διαφέρουν αρκετά μεταξύ τους από άποψη περιεχομένου καθώς είναι επαρκώς μικρές σύμφωνα με τα αποτελέσματα που προέκυψαν για τα σύνολα δεδομένων που χρησιμοποιήθηκαν στο Κεφάλαιο 5. Η απόκλιση της μεθόδου LSA από τη μέθοδο FileTokens συνεπάγεται ότι η πρώτη θεωρείται πιο αποδοτική για τη συγκεκριμένη σύγκριση με αποτέλεσμα να θεωρείται λιγότερο σημαντική η ομοιότητα του αλγορίθμου FunctionTokens, καθώς αυτή μπορεί να προήλθε από παρόμοιες συναρτήσεις με εντελώς διαφορετική χρήση στο κάθε έργο. Η μέθοδος CallTreeProd θεωρεί εντελώς ανόμοια τα δύο έργα λογισμικού θέτοντας την ομοιότητα των δύο έργων ίση με μηδέν. Επομένως, μπορούμε να εξάγουμε με ασφάλεια το συμπέρασμα ότι επρόκειτο για δύο εντελώς ανόμοια αρχεία. Το συμπέρασμα επιβεβαιώνεται εφόσον το ένα έργο αποτελεί υλοποίηση του προβλήματος PacMan ενώ το άλλο αφορά μια στοίβα προβολής για Android εφαρμογές. Τα δύο έργα λογισμικού, δηλαδή, εκτός από την ανομοιότητά τους, αποτελούν υλοποιήσεις διαφορετικών προβλημάτων.

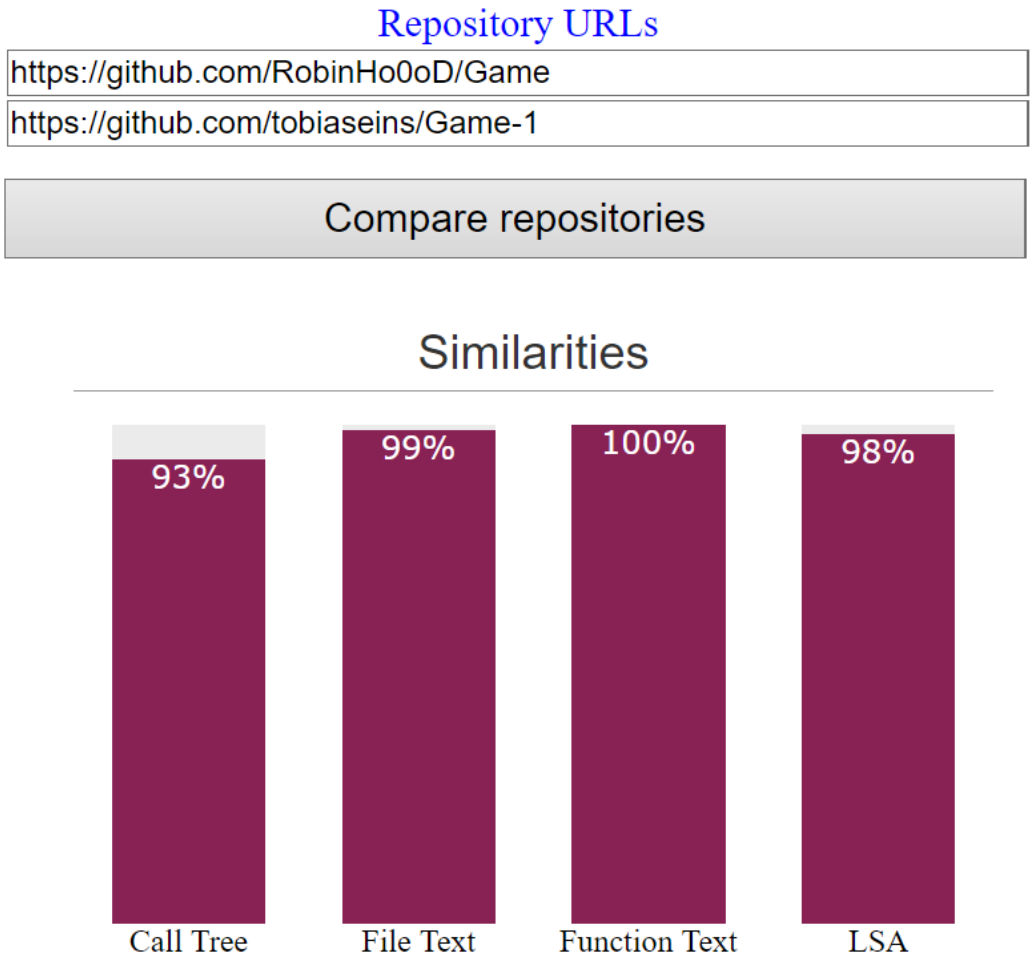


Σχήμα 28: Αποτελέσματα σύγκρισης ανόμοιων έργων

Τέλος, στο Σχήμα 29 συγκρίνουμε δύο έργα λογισμικού τα οποία αποτελούν διακλάδωση του ίδιου έργου. Από τα γραφήματα ράβδων των μεθόδων FileTokens και LSA παρατηρούμε ότι τα αρχεία των δύο έργων λογισμικού είναι πολύ όμοια, ενώ περιέχουν κάποιες μικρές διαφορές. Επίσης, οι συναρτήσεις των έργων είναι πανομοιότυπες μεταξύ τους, όπως φαίνεται από τη μέθοδο FunctionTokens. Ενώ η δομή είναι αρκετά όμοια, εντοπίζονται ορισμένες διαφορές εφόσον η μέθοδος CallTreeProd δίνει ομοιότητα ίση με 93%. Εφόσον η μέθοδος FunctionTokens δίνει τη μέγιστη ομοιότητα μεταξύ των συναρτήσεων, εξάγεται το συμπέρασμα ότι η αλλαγή της ροής του προγράμματος οφείλεται σε λίγες νέες συναρτήσεις που προστέθηκαν και όχι σε αλλαγή των υπαρχόντων. Επομένως, μιλάμε για δύο όμοια αρχεία όπου προστέθηκαν κάποιες παραπάνω λειτουργίες στο ένα.

Τα παραπάνω συμπεράσματα επιβεβαιώνονται αν εξετάσουμε τα έργα λεπτομερώς. Ο αριθμός των αρχείων και στα δύο έργα είναι ίσος, ενώ το ο αριθμός των συναρτήσεων διαφέρουν κατά 5. Οι διαφορετικές συναρτήσεις αυτές είναι μικρής έκτασης και υπάρχουν στα υπάρχοντα αρχεία, οπότε για αυτό και η ομοιότητα των αρχείων αυτών είναι υψηλή αλλά όχι ίση με τη μονάδα. Το πρώτο έργο διαθέτει 8 έργα ενώ το δεύτερο 9. Τα 7 είναι πανομοιότυπα ενώ η ρίζα του όγδοου στο δεύτερο έργο καλεί μια παραπάνω

συνάρτηση σε σχέση με το πρώτο. Από αυτές τις παρατηρήσεις μπορούμε να συμπεράνουμε ότι, πράγματι, τα έργα λογισμικού είναι όμοια με ελάχιστες επεκτάσεις στο ένα.



Σχήμα 29: Αποτελέσματα σύγκρισης όμοιων έργων

7 Συμπεράσματα και μελλοντική εργασία

7.1 Συμπεράσματα

Το αντικείμενο της έρευνάς μας και κατά επέκταση της παρούσας διπλωματικής εργασίας, η εύρεση, δηλαδή, της σημασιολογικής ομοιότητας των έργων λογισμικού, είναι ένα αρκετά απαιτητικό πρόβλημα που θα αποτελεί αντικείμενο έρευνας για πολλά χρόνια ακόμα. Αφότου ορίσαμε το πρόβλημα, αναλύσαμε τις μεθόδους που εφαρμόζονται στην υπάρχουσα βιβλιογραφία και περιγράψαμε τα πιο σημαντικά συστήματα για την κάθε μέθοδο. Εν συνεχεία, περιγράψαμε λεπτομερώς το σύστημα υπολογισμού ομοιότητας που υλοποιήσαμε και αναλαμβάνει να δώσει μια ικανοποιητική λύση στο υπάρχον πρόβλημα.

Όπως και τα υπόλοιπα συστήματα υπολογισμού ομοιότητας, το σύστημά μας δέχεται ως είσοδο ένα σύνολο έργων λογισμικού τα οποία τα συγκρίνει ανά δύο και εξάγει μια τιμή ομοιότητας για την κάθε σύγκριση. Το πλεονέκτημα του συστήματός μας έγκειται στην πολύπλευρη μελέτη του πηγαίου κώδικα για τον υπολογισμό της ομοιότητας. Μελετάται τόσο το περιεχόμενο όσο και η δομή του πηγαίου κώδικα. Για τη μελέτη αυτή, χρησιμοποιείται ένα σύνολο αλγορίθμων που αναλύουν τον κώδικα, δίνοντας ο καθένας από μια πληροφορία σχετικά με την ομοιότητα δύο έργων λογισμικού.

Επίσης, όλοι οι αλγόριθμοι έχουν υψηλή διακριτική ικανότητα. Αυτό, συμβαίνει ακόμα και στην περίπτωση όπου συγκρίνονται έργα λογισμικού των οποίων οι υλοποιήσεις είναι παρόμοιου τύπου αλλά δεν είναι όμοια. Συγκεκριμένα, η μετρική AUC για το πρώτο σύνολο δεδομένων που εφαρμόστηκε κυμαίνεται από 0.905 εώς 0.991, ανάλογα τον αλγόριθμο που εφαρμόζεται. Για ανόμοια σύνολα δεδομένων, η μετρική αυτή λαμβάνει ακόμα υψηλότερες τιμές. Συγκεκριμένα, για το δεύτερο σύνολο δεδομένων όπου τα έργα λογισμικού κάθε φακέλου υλοποιούν πρόβλημα διαφορετικής κατηγορίας, η μετρική AUC ισούται με τη μονάδα, εφαρμόζοντας τον τέλειο διαχωρισμό μεταξύ των έργων.

Επίσης, το σύστημά μας εφαρμόζει όλους τους αλγορίθμους και για την περίπτωση που διατηρούνται τα σχόλια. Από τη μελέτη αυτή, προκύπτει ότι όχι απλά τα σχόλια δεν αποτελούν θόρυβο για τον υπολογισμό της ομοιότητας μεταξύ δύο έργων λογισμικού αλλά οδηγούν σε καλύτερα αποτελέσματα σε σχέση με το αν αφαιρούνταν. Αρχικά, η τιμή ομοιότητας των πραγματικά όμοιων έργων κατά κύριο λόγο αυξάνεται ενώ η τιμή ομοιότητας δύο έργων που ανήκουν σε διαφορετικό φάκελο μειώνεται. Επίσης, τα σχόλια οδηγούν και στον καλύτερο διαχωρισμό των όμοιων έργων από τα ανόμοια κατά την εφαρμογή μεθόδων σύγκρισης του πηγαίου κώδικα ως κείμενο. Η τιμή της μετρικής AUC μιας μεθόδου που αξιοποιεί τα σχόλια είναι υψηλότερη σε σχέση με την τιμή που θα είχε η μετρική αν η ίδια μέθοδος αφαιρούσε τα σχόλια.

7.2 Μελλοντική εργασία

Σε αυτό το σημείο, αναφέρουμε μερικές επεκτάσεις οι οποίες θα μπορούσαν να πραγματοποιηθούν μελλοντικά.

Το σύστημα για κάθε μέθοδο εξάγει και μια τιμή ομοιότητας για δύο έργα λογισμικού. Η μία μέθοδος δεν επικαλύπτει την άλλη, αλλά λειτουργούν συμπληρωματικά ώστε να προσδιοριστούν πλήρως τα σημεία που δύο έργα είναι όμοια αλλά και τα σημεία στα οποία διαφέρουν. Συνεπώς, θα ήταν χρήσιμη η εξαγωγή μιας ενιαίας τιμής ομοιότητας, ο υπολογισμός της οποίας θα βασίζεται στα αποτελέσματα όλων των αλγορίθμων, που θα δηλώνει το ποσοστό ομοιότητας δύο έργων λογισμικού.

Το σύστημά μας υπολογίζει την ομοιότητα έργων λογισμικού, τα οποία υλοποιήθηκαν με τη γλώσσα προγραμματισμού Java. Επομένως, το σύστημα θα μπορούσε να προσαρμοστεί και για άλλες γλώσσες προγραμματισμού, όπως Python ή C++. Συγκεκριμένα, τα εργαλεία που θα έπρεπε να τροποποιηθούν είναι: ο εισαγωγέας δεδομένων που αναλαμβάνει την εισαγωγή των έργων λογισμικού στο σύστημα, ο εξαγωγέας χαρακτηριστικών στο υποσύστημα μελέτης περιεχομένου που αναλαμβάνει την προεπεξεργασία του πηγαίου κώδικα και το διαχωρισμό του σε συναρτήσεις όταν απαιτείται και ο εξαγωγέας δέντρων στο υποσύστημα μελέτης δομής ο οποίος εξάγει τα δέντρα κλήσης συνάρτησης σύμφωνα με τη διάρθρωση που ακολουθεί η γλώσσα προγραμματισμού Java. Οποιοδήποτε άλλο σύστημα είναι ανεξάρτητο της γλώσσας προγραμματισμού των υπό εξέταση έργων λογισμικού και θα μπορούσε να διατηρηθεί.

Μια άλλη προσθήκη η οποία θα βελτίωνε τα αποτελέσματα του συστήματός μας θα ήταν η δυνατότητα χρήσης κάποιου feedback από το χρήστη. Για κάθε σύγκριση έργων λογισμικού ο χρήστης θα μπορούσε να αξιολογεί αν τα έργα λογισμικού είναι όμοια ή όχι. Έτσι, το σύστημα θα μπορούσε να καθορίσει κάποιο κατώφλι ομοιότητας σύμφωνα με το οποίο θα γινόταν και η τελική εκτίμηση για το αν τα δύο αρχεία χαρακτηρίζονται ως όμοια.

Τέλος, το σύστημα θα μπορούσε να περιλαμβάνει ένα γραφικό περιβάλλον που θα το έκανε αρκετά ευχρηστο. Θα μπορούσε να δημιουργηθεί μια ιστοσελίδα όπου θα εισαγόταν τα υπό σύγκριση έργα και θα εξάγονται λίστες όπου θα γίνεται διαχωρισμός των όμοιων από τα ανόμοια έργα λογισμικού και θα αναφέρεται και το ποσοστό ομοιότητας αυτών. Μια άλλη εναλλακτική, θα μπορούσε να είναι η δημιουργία ενός plugin, για παράδειγμα, στον Eclipse.

Α΄ Πίνακες σύγχυσης συνόλου δεδομένων που υλοποιούν παρόμοια ζητήματα

test	A	B	C	D	E	F	G	H	I	J
A	0.974	0.167	0.096	0.169	0.228	0.188	0.097	0.147	0.153	0.144
B	0.167	0.948	0.165	0.295	0.265	0.232	0.126	0.192	0.153	0.139
C	0.096	0.165	0.892	0.258	0.162	0.16	0.0869	0.162	0.108	0.098
D	0.169	0.295	0.258	0.751	0.216	0.226	0.141	0.238	0.188	0.171
E	0.228	0.265	0.162	0.216	1.0	0.327	0.148	0.163	0.168	0.157
F	0.188	0.232	0.16	0.226	0.327	0.997	0.136	0.168	0.169	0.141
G	0.097	0.126	0.087	0.141	0.148	0.136	1.0	0.26	0.144	0.13
H	0.147	0.192	0.162	0.238	0.163	0.168	0.26	0.881	0.152	0.151
I	0.153	0.153	0.108	0.188	0.168	0.169	0.144	0.152	0.995	0.141
J	0.144	0.139	0.098	0.171	0.157	0.141	0.13	0.151	0.141	1.0

Πίνακας 9: Πίνακας σύγχυσης FileTokens+Comments

test	A	B	C	D	E	F	G	H	I	J
A	0.976	0.158	0.094	0.153	0.202	0.193	0.081	0.136	0.156	0.142
B	0.158	0.944	0.184	0.266	0.237	0.219	0.109	0.187	0.154	0.137
C	0.094	0.184	0.895	0.27	0.178	0.192	0.082	0.187	0.119	0.107
D	0.153	0.266	0.27	0.726	0.212	0.241	0.099	0.228	0.169	0.149
E	0.202	0.237	0.178	0.212	0.977	0.327	0.136	0.165	0.18	0.164
F	0.193	0.219	0.192	0.241	0.327	0.996	0.122	0.178	0.186	0.148
G	0.081	0.109	0.0819	0.099	0.136	0.122	1.0	0.244	0.143	0.123
H	0.136	0.187	0.187	0.228	0.165	0.178	0.244	0.868	0.16	0.152
I	0.156	0.154	0.119	0.169	0.18	0.186	0.143	0.16	0.995	0.139
J	0.142	0.137	0.107	0.149	0.164	0.148	0.123	0.152	0.139	1.0

Πίνακας 10: Πίνακας σύγχυσης FileTokens

test	A	B	C	D	E	F	G	H	I	J
A	0.99	0.287	0.17	0.313	0.329	0.326	0.189	0.33	0.237	0.231
B	0.287	0.947	0.272	0.388	0.377	0.393	0.256	0.356	0.254	0.247
C	0.17	0.272	0.99	0.326	0.302	0.312	0.192	0.279	0.216	0.237
D	0.313	0.388	0.326	0.792	0.488	0.505	0.259	0.453	0.392	0.347
E	0.329	0.377	0.302	0.488	0.999	0.468	0.304	0.365	0.311	0.26
F	0.326	0.393	0.312	0.505	0.468	0.997	0.249	0.341	0.327	0.289
G	0.189	0.256	0.192	0.259	0.304	0.249	1.0	0.466	0.299	0.281
H	0.33	0.356	0.279	0.453	0.365	0.341	0.466	0.909	0.317	0.274
I	0.237	0.254	0.216	0.392	0.311	0.327	0.299	0.317	0.999	0.302
J	0.231	0.247	0.237	0.347	0.26	0.289	0.281	0.274	0.302	1.0

Πίνακας 11: Πίνακας σύγχυσης FunctionTokens+Comments

test	A	B	C	D	E	F	G	H	I	J
A	0.99	0.305	0.17	0.315	0.347	0.355	0.195	0.33	0.237	0.231
B	0.304	0.945	0.334	0.409	0.412	0.429	0.256	0.372	0.271	0.254
C	0.17	0.334	0.99	0.327	0.316	0.349	0.201	0.279	0.216	0.237
D	0.315	0.409	0.327	0.791	0.504	0.529	0.263	0.452	0.393	0.349
E	0.347	0.412	0.316	0.504	0.998	0.512	0.328	0.389	0.336	0.272
F	0.355	0.429	0.349	0.529	0.512	0.996	0.269	0.374	0.348	0.307
G	0.195	0.256	0.201	0.26	0.328	0.269	1.0	0.485	0.317	0.29
H	0.33	0.372	0.279	0.452	0.389	0.374	0.485	0.909	0.317	0.274
I	0.237	0.271	0.216	0.393	0.336	0.348	0.317	0.317	0.999	0.302
J	0.231	0.254	0.237	0.349	0.272	0.307	0.29	0.274	0.302	1.0

Πίνακας 12: Πίνακας σύγχυσης FunctionTokens

test	A	B	C	D	E	F	G	H	I	J
A	0.963	0.077	0.041	0.0877	0.088	0.073	0.057	0.081	0.075	0.087
B	0.077	0.884	0.047	0.111	0.085	0.078	0.06	0.072	0.07	0.06
C	0.041	0.047	0.873	0.114	0.043	0.042	0.028	0.045	0.048	0.037
D	0.088	0.111	0.114	0.681	0.091	0.091	0.09	0.097	0.105	0.071
E	0.088	0.085	0.043	0.091	0.999	0.118	0.075	0.072	0.083	0.06
F	0.073	0.078	0.042	0.091	0.118	0.994	0.066	0.07	0.075	0.056
G	0.057	0.06	0.028	0.09	0.075	0.066	1.0	0.124	0.109	0.055
H	0.081	0.072	0.045	0.097	0.072	0.07	0.124	0.87	0.08	0.087
I	0.075	0.07	0.048	0.105	0.083	0.075	0.109	0.08	0.991	0.073
J	0.087	0.06	0.037	0.071	0.06	0.056	0.055	0.087	0.073	1.0

Πίνακας 13: Πίνακας σύγχυσης LSA+Comments

test	A	B	C	D	E	F	G	H	I	J
A	0.961	0.094	0.075	0.099	0.112	0.102	0.062	0.096	0.089	0.094
B	0.094	0.902	0.093	0.128	0.12	0.105	0.067	0.106	0.087	0.08
C	0.075	0.093	0.9	0.19	0.106	0.11	0.05	0.094	0.071	0.062
D	0.099	0.128	0.19	0.671	0.145	0.14	0.094	0.14	0.11	0.084
E	0.112	0.12	0.106	0.145	0.979	0.222	0.116	0.124	0.149	0.106
F	0.102	0.105	0.11	0.14	0.222	0.994	0.132	0.157	0.164	0.109
G	0.061	0.067	0.051	0.094	0.116	0.132	1.0	0.17	0.127	0.074
H	0.096	0.106	0.094	0.14	0.124	0.157	0.17	0.849	0.107	0.117
I	0.089	0.087	0.071	0.11	0.15	0.164	0.127	0.107	0.991	0.073
J	0.094	0.08	0.062	0.084	0.106	0.109	0.074	0.117	0.073	1.0

Πίνακας 14: Πίνακας σύγχυσης LSA

test	A	B	C	D	E	F	G	H	I	J
A	0.387	0.077	0.04	0.137	0.075	0.074	0.049	0.051	0.055	0.081
B	0.077	0.351	0.041	0.169	0.075	0.064	0.05	0.064	0.068	0.061
C	0.04	0.041	0.355	0.216	0.048	0.046	0.033	0.04	0.046	0.048
D	0.137	0.169	0.216	0.625	0.104	0.109	0.147	0.074	0.118	0.099
E	0.075	0.0748	0.048	0.104	0.183	0.076	0.072	0.04	0.058	0.052
F	0.074	0.064	0.046	0.109	0.076	0.21	0.069	0.043	0.063	0.06
G	0.049	0.0504	0.033	0.147	0.072	0.069	0.344	0.094	0.076	0.054
H	0.0513	0.064	0.04	0.074	0.04	0.043	0.094	0.242	0.06	0.082
I	0.055	0.068	0.046	0.118	0.058	0.063	0.076	0.06	0.37	0.074
J	0.081	0.061	0.048	0.099	0.052	0.06	0.054	0.082	0.074	0.363

Πίνακας 15: Πίνακας σύγχυσης LSA+SVD+Comments

test	A	B	C	D	E	F	G	H	I	J
A	0.926	0.0	0.001	0.0	0.0	0.0	0.0	0.0	0.0	0.0
B	0.0	0.389	0.001	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C	0.001	0.001	0.723	0.0	0.001	0.001	0.001	0.001	0.001	0.001
D	0.0	0.0	0.0	0.424	0.0	0.0	0.0	0.0	0.0	0.0
E	0.0	0.0	0.001	0.0	0.919	0.0	0.0	0.0	0.0	0.0
F	0.0	0.0	0.001	0.0	0.0	0.896	0.0	0.0	0.0	0.0
G	0.0	0.0	0.001	0.0	0.0	0.0	1.0	0.0	0.0	0.0
H	0.0	0.0	0.001	0.0	0.0	0.0	0.0	0.648	0.0	0.0
I	0.0	0.0	0.001	0.0	0.0	0.0	0.0	0.0	0.932	0.0
J	0.0	0.0	0.001	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Πίνακας 16: Πίνακας σύγχυσης CallTreeProd+Comments

test	A	B	C	D	E	F	G	H	I	J
A	0.926	0.0	0.001	0.0	0.0	0.0	0.0	0.0	0.0	0.0
B	0.0	0.385	0.001	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C	0.001	0.001	0.723	0.0	0.001	0.001	0.001	0.001	0.001	0.001
D	0.0	0.0	0.0	0.424	0.0	0.0	0.0	0.0	0.0	0.0
E	0.0	0.0	0.001	0.0	0.918	0.0	0.0	0.0	0.0	0.0
F	0.0	0.0	0.001	0.0	0.0	0.89	0.0	0.0	0.0	0.0
G	0.0	0.0	0.001	0.0	0.0	0.0	1.0	0.0	0.0	0.0
H	0.0	0.0	0.001	0.0	0.0	0.0	0.0	0.648	0.0	0.0
I	0.0	0.0	0.001	0.0	0.0	0.0	0.0	0.0	0.932	0.0
J	0.0	0.0	0.001	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Πίνακας 17: Πίνακας σύγχυσης CallTreeProd

test	A	B	C	D	E	F	G	H	I	J
A	0.945	0.023	0.037	0.014	0.023	0.032	0.026	0.027	0.027	0.03
B	0.023	0.641	0.024	0.013	0.034	0.047	0.039	0.036	0.031	0.027
C	0.037	0.024	0.747	0.013	0.024	0.029	0.026	0.026	0.03	0.021
D	0.014	0.013	0.013	0.433	0.03	0.016	0.012	0.014	0.017	0.027
E	0.023	0.034	0.024	0.03	0.928	0.067	0.039	0.026	0.031	0.04
F	0.032	0.047	0.029	0.016	0.067	0.974	0.055	0.041	0.045	0.065
G	0.026	0.039	0.026	0.012	0.039	0.055	1.0	0.062	0.037	0.031
H	0.027	0.036	0.026	0.014	0.026	0.041	0.062	0.781	0.041	0.039
I	0.027	0.031	0.03	0.017	0.031	0.045	0.037	0.041	0.968	0.036
J	0.03	0.027	0.021	0.027	0.04	0.065	0.031	0.039	0.036	1.0

Πίνακας 18: Πίνακας σύγχυσης CallTreeSum+Comments

test	A	B	C	D	E	F	G	H	I	J
A	0.945	0.025	0.037	0.014	0.024	0.031	0.023	0.027	0.027	0.03
B	0.025	0.644	0.027	0.015	0.026	0.047	0.031	0.037	0.032	0.032
C	0.037	0.027	0.747	0.013	0.023	0.029	0.024	0.026	0.03	0.021
D	0.014	0.015	0.013	0.433	0.037	0.017	0.011	0.014	0.017	0.027
E	0.024	0.026	0.023	0.037	0.926	0.076	0.031	0.03	0.033	0.046
F	0.031	0.047	0.029	0.017	0.076	0.973	0.053	0.041	0.046	0.066
G	0.023	0.031	0.024	0.011	0.031	0.053	1.0	0.065	0.036	0.033
H	0.027	0.037	0.026	0.014	0.03	0.041	0.065	0.781	0.041	0.039
I	0.027	0.032	0.03	0.017	0.033	0.046	0.036	0.041	0.968	0.036
J	0.03	0.032	0.021	0.027	0.046	0.066	0.033	0.039	0.036	1.0

Πίνακας 19: Πίνακας σύγχυσης CallTreeSum

test	A	B	C	D	E	F	G	H	I	J
A	0.859	0.003	0.004	0.005	0.004	0.004	0.004	0.003	0.004	0.005
B	0.003	0.409	0.003	0.003	0.005	0.007	0.006	0.003	0.004	0.004
C	0.004	0.003	0.696	0.004	0.004	0.004	0.003	0.003	0.004	0.003
D	0.005	0.003	0.004	0.427	0.011	0.004	0.004	0.006	0.006	0.014
E	0.004	0.005	0.004	0.011	0.908	0.009	0.007	0.004	0.005	0.008
F	0.004	0.007	0.004	0.004	0.009	0.923	0.009	0.006	0.006	0.011
G	0.004	0.006	0.003	0.004	0.007	0.009	1.0	0.009	0.005	0.006
H	0.003	0.003	0.003	0.006	0.004	0.006	0.009	0.68	0.007	0.006
I	0.004	0.004	0.004	0.006	0.005	0.006	0.005	0.007	0.932	0.006
J	0.005	0.004	0.003	0.014	0.008	0.011	0.006	0.006	0.006	1.0

Πίνακας 20: Πίνακας σύγχυσηςCallTreeHeur+Comments

test	A	B	C	D	E	F	G	H	I	J
A	0.859	0.004	0.004	0.006	0.004	0.004	0.004	0.003	0.004	0.005
B	0.004	0.408	0.003	0.003	0.003	0.006	0.005	0.004	0.004	0.005
C	0.004	0.003	0.696	0.004	0.003	0.003	0.003	0.003	0.004	0.003
D	0.006	0.003	0.004	0.427	0.012	0.004	0.004	0.006	0.006	0.015
E	0.004	0.003	0.003	0.012	0.908	0.009	0.004	0.005	0.005	0.01
F	0.004	0.006	0.003	0.004	0.009	0.916	0.009	0.006	0.006	0.011
G	0.004	0.005	0.003	0.004	0.004	0.009	1.0	0.01	0.006	0.006
H	0.003	0.004	0.003	0.006	0.005	0.006	0.01	0.68	0.007	0.006
I	0.004	0.004	0.004	0.006	0.005	0.006	0.006	0.007	0.932	0.006
J	0.005	0.005	0.003	0.015	0.01	0.011	0.006	0.006	0.006	1.0

Πίνακας 21: Πίνακας σύγχυσης CallTreeHeur

Β΄ Πίνακες σύγχυσης συνόλου δεδομένων που υλοποιούν διαφορετικά ζητήματα

test	A	B	C	D
A	0.932	0.0	0.0	0.0
B	0.0	0.625	0.0	0.0
C	0.0	0.0	0.696	0.0
D	0.0	0.0	0.0	0.607

Πίνακας 22: Πίνακας σύγχυσης CallTreeProd+Comments

test	A	B	C	D
A	0.991	0.136	0.078	0.062
B	0.136	0.929	0.182	0.082
C	0.078	0.182	0.966	0.079
D	0.062	0.082	0.079	0.957

Πίνακας 23: Πίνακας σύγχυσης LSA

test	A	B	C	D
A	0.995	0.140	0.137	0.07
B	0.14	0.963	0.239	0.093
C	0.137	0.239	0.994	0.124
D	0.07	0.093	0.124	0.976

Πίνακας 24: Πίνακας σύγχυσης FileTokens

test	A	B	C	D
A	0.999	0.212	0.265	0.18
B	0.212	0.979	0.323	0.174
C	0.265	0.323	0.993	0.168
D	0.18	0.174	0.168	0.985

Πίνακας 25: Πίνακας σύγχυσης FunctionTokens

References

[1] R. Curley, "Architects of the information age.," pp. 104–105, 2011.

[2] M. D. McIlroy, "Mass-produced software components.," *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany.*, 1968.

[3] W. B. Frakes and K. Kang, "Software reuse research: Status and future.," *IEEE transactions on Software Engineering 31.7*, pp. 529–536, 2005.

[4] GitHub, "About github." `https://github.com/about`.

[5] Williams, "Github pours energies into enterprise – raises $100 million from power vc andreessen horowitz." `https://techcrunch.com`, 7 2012.

[6] LosTechies, "Why github's pricing model stinks (for us)." https://lostechies.com, 7 2012.

[7] Wired, "The problem with putting all the world's code in github." https://www.wired.com, 6 2015.

[8] SourceForge, "About sourceforge." `https://sourceforge.net/about`.

[9] StackOverflow, "About stackoverlfow." `https://stackoverflow.com/company`.

[10] J. Atwood, "The gamification." `https://blog.codinghorror.com`, 9 2008.

[11] S. Overflow, "What is reputation? how do i earn (and lose) it?." `https://stackoverflow.com/help/whats-reputation`, 8 2010.

[12] I. L. Cynthia Kustanto, "Automatic source code plagiarism detection," *10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*, vol. 1, no. 1, pp. 481–482, 2009.

[13] A. E. Ameera Jadalla, "Pde4java: Plagiarism detection engine for java source code: A clustering approach," *International Journal Business Intelligence and Data Mining*, vol. 3, no. 2, pp. 121–135, 2008.

[14] A. Mockus, "Large-scale code reuse in open source software," *First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS'07: ICSE Workshops 2007)*, 2007.

[15] P. S. K. X. X. Q. L. Yun Zhang, David Lo and J. Sun, "Detecting similar repositories on github," *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, vol. 1, no. 1, pp. 13–23, 2017.

[16] V. D. K. A. N. Sandhya, Y. Sri Lalitha and D. A. Govardhan, "Analysis of stemming algorithm for text clustering," *International Journal of Computer Science Issues*, vol. 8, no. 1, pp. 1–8, 2011.

[17] V. R. Andrian Marcus, Andrey Sergeyev and J. I. Maletic, "An information retrieval approach to concept location in source code," *11th Working Conference on Reverse Engineering*, pp. 214–223, 2004.

[18] G. Cosma and M. Joy, "An approach to source-code plagiarism detection and investigation using latent semantic analysis," *IEEE TRANSACTIONS ON COMPUTERS*, vol. 61, no. 3, pp. 379–394, 2012.

[19] J. W. PENG WU and B. TIAN, "Software homology detection with software motifs based on function-call graph," *IEEE Access*, vol. 6, no. 7, pp. 19007–19017, 2018.

[20] H. Y. Deqiang Fu, Yanyan Xu and B. Yang, "Wastk: A weighted abstract syntax tree kernel method for source code plagiarism detection," *Scientific Programming*, vol. 2017, no. 1, pp. 1–8, 2017.

[21] G. R. Michel Chilowicz, Etienne Duris, "Syntax tree fingerprinting for source code similarity detection," *IEEE 17th International Conference on Program Comprehension*, vol. 1, no. 1, pp. 243–247, 2009.

[22] K. X. Jianglang Feng, Baojiang Cui, "A code comparison algorithm based on ast for plagiarism detection," *2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies*, vol. 1, no. 1, pp. 393–397, 2013.

[23] Q. H. Guo Tao, Dong Guowei and C. Baojiang, "Improved plagiarism detection algorithm based on abstract syntax tree," *2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies*, vol. 1, no. 1, pp. 714–719, 2013.

[24] T. K. Maxim Mozgovoy and G. Cosma, "Automatic student plagiarism detection: future perspectives," *EDUCATIONAL COMPUTING RESEARCH*, vol. 43, no. 4, pp. 511–531, 2010.

[25] M. Berry and M. Browne, *Understanding Search Engines: Mathematical Modeling and Text Retrieval (Software, Environments, Tools), Second Edition*. Wiley, 2005.

[26] T. L. S. D. S. Dumais, G. Furnas and R. Harshman, "Using latent semantic analysis to improve access to textual information," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 281–285, 1988.

[27] S. D. M. Berry and G. O'Brien, "Using linear algebra for intelligent information retrieval," *Technical Report UT-CS-94-270, University of Tennessee Knoxville, TN, USA*, 1994.

[28] T. L. S. D. S. Dumais, G. Furnas and R. Harshman, "A framework for understanding latent semantic indexing (lsi) performance," *Information Processing and Management*, vol. 42, no. 1, pp. 56–73, 2006.

[29] Z. Duric and D. Gasevic, "A source code similarity system for plagiarism detection," *The Computer Journal*, vol. 56, no. 1, pp. 70–86, 2013.

[30] M. J. Wise, "Yap3: Improved detection of similarities in computer program and other texts," *Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education*, vol. 6, no. 7, pp. 130–134, 2018.

[31] G. M. Lutz Prechelt and M. Phlippsen, "Finding plagiarisms among a set of programs with jplag," *Journal of Universal Computer Science*, vol. 8, no. 11, pp. 1016–1038, 2002.

[32] M. J. Wise, "String similarity via greedy string tiling and running karp-rabin matching," *Online Preprint, Dec*, vol. 119, 1993.

[33] P. Willett, "The porter stemming algorithm: then and now," *Program: electronic library and information systems*, vol. 40, no. 3, pp. 219–223, 2006.

[34] P. Willett, "An algorithm for suffix stripping," *Program: electronic library and information systems*, vol. 14, no. 3, pp. 130–137, 1980.

[35] Lovins, "Development of a stemming algorithm," *Mechanical Translation and Computational Linguistics*, vol. 11, no. 1/2, pp. 22–31, 1965.

[36] G. SALTON and C. BUCKLEY, "Term-weighting approaches in automatic text retrieval," *Information Processing and Management*, vol. 24, no. 5, pp. 513–523, 1988.

[37] W. YANG, "Identifying syntactic differences between two programs," *SOFTWARE-PRACTICE AND EXPERIENCE*, vol. 21, no. 7, pp. 739–755, 1991.

[38] H.-J. S. S.-B. P. Jeong-Woo Son, Tae-Gil Noh, "An application for plagiarized source code detection based on a parse tree kernel," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 8, pp. 1911–1918, 2013.

[39] M. J. Wise, "Detection of similarities in student programs: Yap'ing maybe preferable to plague'ing," *ACM SIGCSE Bulletin*, vol. 24, no. 1, pp. 268–271, 1992.

[40] A. Moschitti, "Making tree kernels practical for natural language learning," pp. 113–120, 2006.