HistoRank: History-Based Ranking of Co-change Candidates

Manishankar Mondal Banani Roy Chanchal K. Roy Kevin A. Schneider
Department of Computer Science, University of Saskatchewan, Saskatoon, Canada
{mshankar.mondal, banani.roy, chanchal.roy, kevin.schneider}@usask.ca

***Abstract*—Evolutionary coupling is a well investigated phenomenon during software evolution and maintenance. If two or more program entities co-change (i.e., change together) frequently during evolution, it is expected that the entities are coupled. This type of coupling is called evolutionary coupling or change coupling in the literature. Evolutionary coupling is realized using association rules and two measures: support and confidence. Association rules have been extensively used for predicting co-change candidates for a target program entity (i.e., an entity that a programmer attempts to change). However, association rules often predict a large number of co-change candidates with many false positives. Thus, it is important to rank the predicted co-change candidates so that the true positives get higher priorities.**

The predicted co-change candidates have always been ranked using the support and confidence measures of the association rules. In our research, we investigate five different ranking mechanisms on thousands of commits of ten diverse subject systems. On the basis of our findings, we propose a history-based ranking approach, *HistoRank (History-based Ranking)*, that analyzes the previous ranking history to dynamically select the most appropriate one from those five ranking mechanisms for ranking co-change candidates of a target program entity. According to our experiment result, *HistoRank* outperforms each individual ranking mechanism with a significantly better MAP (mean average precision). We investigate different variants of HistoRank and realize that the variant that emphasizes the ranking in the most recent occurrence of co-change in the history performs the best.

***Index Terms*—Evolutionary Coupling, Association Rules, Co-change Candidates, Ranking**

I. Introduction

Evolutionary coupling, also known as logical coupling or change coupling, has been extensively investigated in software engineering research and practice. If a group of program entities changed together (i.e., co-changed) frequently during the past evolution of a software system, it is expected that the entities in the group are coupled. This type of coupling which we can realize from the evolutionary history of a software system is called evolutionary coupling. If a group of program entities exhibited evolutionary coupling in the past, a change in one entity in the future is likely to require corresponding changes to the other entities in the group.

A great many studies [1], [3], [5], [7], [8], [10], [11], [13], [14], [19], [21], [22], [25], [27], [31] have been conducted on the detection and usage of evolutionary coupling. While evolutionary coupling has been used for identifying buggy program entities [3], detecting software design deficiencies [10], requirements traceability, refactoring and tracking of code clones [15], [16], and model driven software development [22], it has been primarily been used for predicting co-change candidates of program entities [5], [9], [13], [25].

When a programmer attempts to make changes to a particular program entity which we call the target entity, the other entities that are related to it (i.e., coupled with it) might get impacted. These other entities constitute the impact set of the target entity. Evolutionary coupling has been commonly used for predicting this impact set. The entities in the impact set of a target entity are called the co-change candidates of the target entity, because these co-change candidates might need to be changed together with the target entity in order to ensure consistency of the software system.

Existing studies [25], [27] have realized evolutionary coupling using association rules [20] with two related measures: support and confidence. While support is the co-change frequency (i.e., how many times two or more entities co-changed) of program entities, confidence is a conditional co-change probability measured using support. We will discuss association rules, support, and confidence in Section 2.

The existing studies and techniques have mined co-change candidates (i.e., the impact set) of a target program entity by analyzing association rules. However, the impact set obtained from the association rules is often very large with many false positives (i.e., the members that do not actually get impacted for changing the target entity). Thus, ranking the members in the impact set is essential so that the true positives (i.e., the members that actually need to be co-changed with the target program entity) get higher priorities. Without proper ranking of the co-change candidates, programmers might often get overwhelmed while identifying the likely co-change candidates from a long list of suggestions obtained from the association rules.

The co-change candidates of the target program entities have always been ranked using the support and confidence measures of the association rules [15], [25], [27]. However, our investigation shows that ranking using support and confidence might not always provide good results. Sometimes, better ranking of the co-change candidates can be achieved by using different other ranking mechanisms such as recency ranking or similarity ranking. In our research, we particularly investigate five ranking mechanisms: (1) support ranking, (2) confidence ranking, (3) recency ranking, (4) similarity ranking, and (5) file proximity ranking for ranking the co-change candidates of the target program entities. From our investigation on thousands of commits of 10 subject systems written in Java, C, and C#, we observe that different ranking mechanisms might

TABLE I
RESEARCH QUESTIONS

RQ	Research Question
RQ 1	Is it important to rank the co-change candidates retrieved using evolutionary coupling?
RQ 2	What is the comparative scenario among different ranking mechanisms for ranking co-change candidates retrieved using evolutionary coupling?
RQ 3	Can we select a suitable ranking mechanism on the basis of the past history for ranking co-change candidates?
RQ 4	Which variant of history based ranking performs the best in ranking co-change candidates?

appear to be promising for ranking co-change candidates of different target entities. We finally realize that a single ranking mechanism should not be considered always for ranking co-change candidates during evolution. Focusing on this, we propose a ranking approach, HistoRank, that analyzes the past co-change history for selecting the best ranking mechanism from the aforementioned ranking mechanisms for ranking the co-change candidates of a target program entity. We answer four research questions listed in Table I through our research. According to our investigation and analysis:

(1) Our proposed ranking approach, HistoRank, outperforms all the five ranking mechanisms (mentioned above) with a significantly higher MAP (mean average precision) in ranking co-change candidates.

(2) From our investigation on different variants of HistoRank we observe that the variant that emphasizes the ranking in the most recent occurrence of co-change in the past performs the best in ranking co-change candidates.

We believe that our proposed ranking approach can be very useful to the programmers in identifying the likely co-change candidates from a long list of co-change suggestions in a time efficient manner. The implementation and data from our research are available online [32].

The rest of the paper is organized as follows. Section II describes the terminology, Section III discusses the ranking mechanisms with necessary examples, Section IV explains our experimental setup and steps, Section V and VI answer our research questions by presenting and discussing our experimental results, Section VII discusses the findings from our experimental results, Section VIII describes the possible threats to validity, Section IX discusses the related work, and finally, Section X presents the concluding remarks.

II. TERMINOLOGY

Evolutionary coupling among program entities is realized using association rules. We define an association rule in the following way according to the literature [20], [27].

Association Rule. An association rule [20] is an expression of the form $X => Y$, where X is the antecedent and Y is the consequent. Each of X and Y is a set of one or more program entities. Such an association rule means that if X gets changed in a commit operation, Y also has a possibility of getting changed in that commit operation. An association rule has two measures: **support** and **confidence**.

Support is the number of commits in which an entity or a group of entities changed together [27]. Let us assume two program entities E_1 and E_2. E_1 changed in the commits: 5, 6, 10, and 24. E_2 changed in the commit operations: 2, 6, 24, 34, and 50. Thus, the support of E_1 is 4 and the support of E_2 is 5. Moreover, the support of $E_1 \cup E_2$ is 2 because E_1 and E_2 together changed in two commits: 6 and 24. The support of a rule, $X => Y$, is calculated in the following way.

$$support(X => Y) = support(X \cup Y) \quad (1)$$

The **confidence** of an association rule $X => Y$ is the conditional probability that Y will change in a commit operation given that X has changed in that commit operation. We calculate confidence using the following equation.

$$confidence(X => Y) = support(X \cup Y)/support(X) \quad (2)$$

If we consider the previous example of the entities E_1 and E_2, we can make two association rules, $E_1 => E_2$ and $E_2 => E_1$, from these. Now, confidence ($E_1 => E_2$) = support ($E_1 \cup E_2$) / support (E_1) = 2/4 = 0.5 In the same way, confidence ($E_2 => E_1$) = 2/5 = 0.4.

In our experiment, we consider method level association rules. In such rules, the antecedents and consequents are methods.

III. RANKING MECHANISMS

This section describes the ranking mechanisms that we investigate in our research.

A. Support Ranking

Rolfsness et al. [25] used this ranking mechanism for ranking file level co-change candidates retrieved using evolutionary coupling. In this mechanism, the co-change candidates of a target program entity are ordered according to the support values of the association rules that they make with the target entity. Co-change candidates with higher support values are given higher priorities. Two or more co-change candidates with the same support value are ordered according to their confidence values. In this case, the candidates with higher confidence values get higher priorities. We explain this with a simple example.

Example. Let us assume that a programmer is going to make changes to a target program entity E1. In order to ensure consistency of the software system, some other entities might also need to be co-changed (i.e., changed together) with E1. By applying Tarmaq algorithm on the past evolutionary history of the program entities (as was done by Rolfsness et al. [25]), we obtain a set of co-change candidates for E1. Let us assume that this set for E1 consists of the entities: E2, E3, E4, and E5. We denote this set by SCC (set of suggested co-change candidates). Each of the entities in the set SCC makes an association rule with E1 where E1 is the antecedent and the entity in the set is the consequent. As a result, we get four association rules: E1=>E2, E1=>E3, E1=>E4, and E1=>E5 in total. We find the supports and confidences of these association rules using the equations Eq. 1 and 2. Let us assume that the supports and confidences of these association rules are: 2, 4, 6, 4 and 0.2, 0.1, 0.3, 0.4 respectively. On the basis of the support values, the co-change candidates can be ordered as follows: E4, E3, E5, E2. We see that the association rules where E3 and E5 are the consequents have the same support value (4). Thus, E3 and E5 appear together after ordering by the support values. We now try to reorder these E3 and E5 considering the confidence values. We see that the rule E1=>E5 has a higher confidence than the rule E1=>E3. Thus, E5 is given a higher priority than E3. The final ordering of the co-change candidates is: E4, E5, E3, E2.

B. Confidence Ranking

This ranking mechanism is just the opposite of the support ranking mechanism (Section III-A). We first order the co-change candidates on the basis of the confidence values of the rules that they make with the target entity. The candidates with higher confidence values are given higher priorities. The candidates with the same confidence values are reordered according to the support values. The candidates with higher supports get higher priorities.

C. Recency Ranking

In this ranking mechanism, the co-change candidates are ranked/ordered on the basis of how recently they co-changed with the target program entity. A co-change candidate that co-changed with the target program entity more recently is given a higher priority in this mechanism.

Example. Let us again assume that a programmer is going to make changes to a target program entity E1. The co-change candidates (for E1) that we retrieved by analyzing the past evolutionary history of the software system using Tarmaq algorithm are E2, E3, E4, and E5. For each of these co-change candidates, we determine the last commit operation where it co-changed with the target entity. Let us assume that the co-change candidate, E2, co-changed with E1 in two commit operations: 50 and 56. Thus, 56 is the last commit operation where E2 co-changed with E1. This last commit operation is denoted as the recency of E2. We determine the recency of each of the co-change candidates of E1. Let us assume that the recency values of the four co-change candidates (E2, E3, E4, and E5) are 56, 34, 56, and 67 respectively. We order the candidates on the basis of these recency values. A co-change candidate with a higher recency gets a higher priority. Thus, the ordering of the co-change candidates according to recency ranking is as follows: E5, E2, E4, and E3. Here, E5 has the highest priority because its recency is the highest (67). We see that the two entities E2 and E4 have the same recency (56). We try to reorder these entities on the basis of the support values of the rules that they make with the target entity. Let us assume that the association rules, E1=>E2 and E1=>E4, have the support values of 2 and 6 respectively. In other words, E4 co-changed with E1 in a higher number of commit operations than with E2. In this situation, E4 gets a higher priority than E2. Thus, the final ordering of the co-change candidates according to recency ranking is: E5, E4, E2, and E3.

TABLE II
SUBJECT SYSTEMS

Systems	**Lang.**	**Domains**	**LLR**	**Revs**
Ctags	C	Code Definition Generator	33,270	774
Camellia	C	Multimedia	85,015	207
Carol	Java	Game	25,091	1,700
Freecol	Java	Game	91,626	1,950
jEdit	Java	Text Editor	1,91,804	4,000
Jabref	Java	Reference Manager	45,515	1,545
MonoOSC	C#	Formats and Protocols	18,991	315
DNSJava	Java	DNS Protocol	23,373	1,635
GreenShot	C#	Multimedia	37,628	999
SQLBuddy	C#	MSDE / Sql Server IDE	18,387	945

LLR = LOC in the Last Revision Revs = No. of Revisions

D. File-proximity Ranking

In this ranking mechanism, the co-change candidates of a target program entity are ranked on the basis of their file-level distances from the target entity in the file system tree. Candidates with lower distances are given higher priorities. Candidates with the same distance are reordered on the basis of the support values of the association rules that they make with the target program entity.

E. Lexical Similarity Ranking

In this ranking mechanism, the co-change candidates are ranked on the basis of their similarity with the target entity. Candidates with higher similarity are given higher priorities. In our research, the program entities are methods. In order to quantify the similarity between a target method and its co-change candidate, we determine the Dice–Sørensen co-efficient [12], [26] between the methods.

Calculating Dice–Sørensen co-efficient between two methods. From each method we obtain a string by appending the source code lines in the method serially one after another. We determine Dice–Sørensen co-efficient between two methods in the following way from the two strings obtained from the methods.

Dice–Sørensen co-efficient: Dice–Sørensen coefficient (DiSC) [12], [23], [26] measures the lexical similarity of any two given strings from their bigrams (the set of every sequence of two adjacent characters) according to Eq. 3:

$$DiSC = \frac{2 \times |bigrams(str_1) \cap bigrams(str_2)|}{|bigrams(str_1)| + |bigrams(str_2)|} \quad (3)$$

Here, $DiSC$ is the Dice–Sørensen co-efficient between the two strings: str_1 and str_2. The Dice–Sørensen coefficient reports similarity values in the range 0 and 1, which we express as percentages. A similarity value of 100% indicates that the strings are the same, whereas 0% indicates that the strings are dissimilar. Dice–Sørensen coefficient rewards both common substrings and a common ordering of those substrings. It is robust to changes in word order. It is language indepedent and it outperforms the existing algorithms, such as Soudex Algorithm, Edit Distance, and Longest Common Subsequence in determining lexical similarity between strings [24], [35].

IV. Experiment setup and steps

This section first discusses the setup of our experiment and then elaborates on the experimental steps.

Subject Systems. We conduct our investigation on ten subject systems downloaded from an on-line repository called SourceForge.net [33]. Table II shows the details of our subject systems. We download each of the revisions of the systems as mentioned in Table II for our analysis. We see that the subject systems are of diverse variety in terms of their application domains, sizes, revision history lengths, and implementation languages. We select the systems in our experiment by focusing on their diversity, because we wanted to generalize our findings regarding ranking co-change candidates retrieved using evolutionary coupling.

Co-change Suggestion Technique. We select the co-change suggestion technique called Tarmaq [25] in our research because it is a promising technique based on mining association rules. It significantly outperforms the existing techniques called Rose and SVD [25] because it can suggest co-change candidates even for previously unseen queries. Pugh et al. [36] proposed Atari which can provide co-change suggestions by mining association rules through an adaptive approach. They showed that Atari can achieve an accuracy level like Tarmaq using a dynamically selected smaller number of commit transactions. Islam et al. [13] proposed transitive association rules which can provide co-change suggestions among entities that did not exhibit evolutionary coupling in the past. However, Islam et al.'s technique is very time-consuming, and thus, it is not suitable for making co-change suggestions in real-time coding environment. Also, the technique often provides a lower precision than Tarmaq [13]. We finally decide that Tarmaq is a reasonable choice for our research.

Experimental steps. We conduct a number of experimental steps sequentially for each of our subject systems before analyzing the ranking mechanisms. We describe these steps in the following way.

- **Downloading revisions.** We download all the revisions as indicated in Table II from the on-line repository.
- **Change detection.** We detect source code changes between every two consecutive revisions by applying UNIX *diff* operation. This operation provides three types of changes: additions, deletions, and modifications with line numbers. The database of changes that we detect for our subject systems is available online [32].
- **Method detection.** We detect methods from each of the revisions by applying a tool called CTAGS [4]. The methods are detected along with their container file paths and starting and ending line numbers.
- **Change mapping.** We map the source code changes that occurred in each revision to the methods of the corresponding revision. Change mapping is particularly done by matching the file paths, and starting and ending line numbers of the changes and methods.
- **Method genealogy detection.** We detect method genealogies considering the methods detected from all the revisions by applying the technique proposed by Lozano and Wermelinger [2].
- **Applying Tarmaq.** We apply the Tarmaq algorithm proposed by Rolfsness et al. [25] for detecting impact sets (co-change candidates) for target methods. Tarmaq suggests impact sets by analyzing the past evolutionary history of the methods.
- **Investigating ranking.** We finally rank the co-change candidates (i.e., the members in the impact sets) by applying different ranking mechanisms and compare these ranking mechanisms by calculating MAP (mean average precision).

We will describe the last step in Section V.

A. Detecting method genealogies

In order to provide co-change suggestions considering method level granularity, we detect the method genealogies from the entire evolutionary history of our subject systems. We define a method genealogy in the following way.

Method Genealogy. Let us assume that a particular method was created in a particular revision of a subject system. The method was alive in n consecutive revisions. Thus, each of these n revisions has a snapshot of the method. The genealogy of the method consists of its n snapshots from the n revisions where it was alive. By detecting the genealogy of a method, we can easily determine how it evolved over the evolution.

Method genealogy detection. We detect method genealogies in our research for realizing how they co-evolved over the evolution. Method genealogy detection was performed by applying the technique which was proposed by Lozano and Wermelinger [2]. The databases containing the method genealogies from our subject systems are available on-line [32]. The database for each subject system contains data regarding which methods were changed together in which commit operations.

B. Applying Tarmaq for retrieving impact sets considering method level granularity

The Tarmaq algorithm was introduced by Rolfsness et al. [25]. It takes a set of entities as the query and analyzes the past entity co-change history to retrieve co-change candidates for the entity set. In our research, we provide a target method as the query to the Tarmaq algorithm. It then analyzes the past method co-change history to find the impact set (i.e., the co-change candidates) of the target method. As we detect method genealogies, we can determine which methods co-changed in which commit operations. Tarmaq uses this method co-change information for getting the co-change candidates.

V. Experiment Results and Analysis

We apply our implementation on each of our subject systems and obtain experiment results. We analyze these results for answering our research questions listed in Table I.

TABLE III
NUMBER OF PREDICTED, ACTUAL, AND TRUE POSITIVE CO-CHANGE CANDIDATES

Subject systems	Predicted	Actual	True Positives
Ctags	4,677	1,568	894
Camellia	6,321	1,612	684
Carol	7,512	7,404	2,084
Freecol	44,308	12,566	3,070
jEdit	10,445	7,100	768
Jabref	1,51,150	25,446	8,914
MonoOSC	25,829	5,754	2,612
DNSJava	52,626	17,160	6,928
GreenShot	26,472	7,478	2,636
SqlBuddy	31,804	8,362	2,964

Predicted = No. of all predicted co-change candidates from all the target methods from entire evolution.
Actual = No. of all actual co-change candidates for all target methods.
True Positives = No. of all true positives for all target methods.

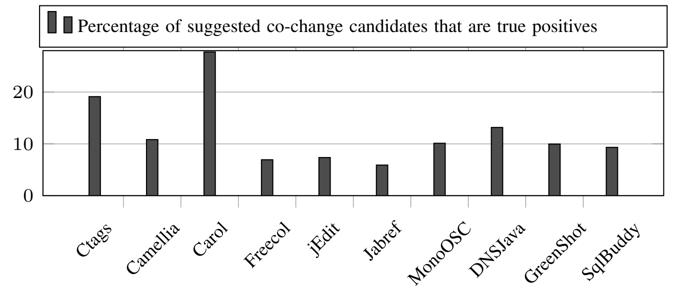

Fig. 1. Percentage of suggested co-change candidates that are true positives

A. Answering RQ 1

RQ 1: *Is it important to rank the co-change candidates retrieved using evolutionary coupling?*

Motivation. Before analyzing and comparing the ranking mechanisms, we wanted to realize whether it is really important to rank the co-change candidates retrieved by using evolutionary coupling. If we observe that most of the retrieved co-change candidates for a target program entity are true positives (i.e., the candidates really need to be co-changed with the target), ranking of the candidates might not be important. However, ranking can be important if only a few of the suggested (i.e., retrieved) co-change candidates are true positives. A good ranking mechanism can assume high priorities for those few true positives so that programmers can easily identify those from a long list of suggested co-change candidates. We answer RQ 1 in the following way.

Experiment procedure. We examine each of the commit operations sequentially from the beginning one. Let us assume that a particular commit operation c was applied on a particular revision r and n methods in that revision were changed. If we consider one of these n methods as the target method, the other $n-1$ methods are the actually co-changed candidates of the target method. However, we want to determine how many of these actually co-changed candidates can be retrieved by analyzing the past evolutionary history of the software system. The past evolutionary history consists of the commit operations: 1 to $c-1$. We apply the Tarmaq algorithm on this past history in order to suggest (i.e., retrieve) co-change candidates for the target method. Tarmaq suggests co-change candidates by analyzing the evolutionary coupling among methods. We should note that in the original study conducted by Rolfsness et al. [25], Tarmaq was applied for analyzing file level evolutionary coupling. In our study, we apply this algorithm for analyzing method level evolutionary coupling for suggesting method level co-change candidates. We use Tarmaq algorithm in our study, because it shows a higher precision and recall than ROSE and SVD [25]. Moreover, Tarmaq can retrieve co-change candidates for previously unseen queries.

Let us consider a target method m in the commit operation c. The set of suggested co-change candidates retrieved by Tarmaq for m from the past commits (1 to $c-1$) is SCC (Suggested Co-change Candidates). As we examine the commit operation c, we determine the set of methods that actually co-changed with m in c. We denote this set of actually co-changed candidates by ACC (Actually Co-changed Candidates). We realize that from the intersection of the two sets, SCC and ACC, we get those suggested co-change candidates that actually co-changed with m in commit c. These candidates (obtained from the intersection of ACC and SCC) are the true positive suggestions.

By considering each of the methods changed in each of the commit operations, we determine the three sets: SCC, ACC, and SCC $\cap$ ACC. We determine the corresponding summations of the number of elements in these sets. Thus, for three sets, we obtain three summations. The values of these summations for each of our candidate subject system are listed in Table III. We determine the percentage of true positives in suggested co-change candidates using the following equation.

$$PTP = \frac{|SCC \cap ACC| \times 100}{|SCC|} \quad (4)$$

Here, PTP is the percentage of true positives in the suggested co-change candidates.

Fig. 1 shows the value of PTP for each of our subject systems. From the figure, we see that the percentage of true positives in the suggested co-change candidates is very low for each subject system. The system called Carol shows the highest percentage which is around 28%. For five out of ten subject systems, the percentage of true positives is less than 10%. Such a scenario implies that most of the suggested co-change candidates are false positives. Thus, ranking of co-change candidates suggested by evolutionary coupling is important so that the true positives get high priorities. Otherwise, programmers might get overwhelmed with too many false positives when identifying true positives from a long list of suggestions. We finally answer RQ 1 in the following way.

Answer to RQ 1. Our analysis establishes the fact that ranking co-change candidates suggested using evolutionary coupling is important, because the percentage of true positives in the suggested (predicted) co-change candidates is generally very low.

TABLE IV
MAPs PROVIDED BY DIFFERENT RANKING MECHANISMS

Systems	S-Ranking	R-Ranking	Sim-Ranking	FP-Ranking	C-Ranking
Ctags	0.3511	0.3730	0.3497	0.3651	0.3275
Camellia	0.2487	0.2815	0.2683	0.2605	0.2123
Carol	0.2279	0.2294	0.2484	0.2511	0.2151
Freecol	0.1496	0.1554	0.1420	0.1767	0.1211
jEdit	0.0497	0.0511	0.0597	0.0543	0.0453
Jabref	0.1708	0.1664	0.1530	0.1922	0.1275
MonoOSC	0.2226	0.2334	0.2226	0.2583	0.1707
DNSJava	0.2744	0.2874	0.2738	0.2625	0.2313
Greenshot	0.2099	0.2191	0.2246	0.2322	0.1740
Sqlbuddy	0.2183	0.2153	0.2158	0.2496	0.1854

S-Ranking = Support Ranking Mechanism
R-Ranking = Recency Ranking Mechanism
Sim-Ranking = Similarity Ranking Mechanism
FP-Ranking = File-proximity Ranking Mechanism
C-Ranking = Confidence Ranking Mechanism

Such a finding inspires us to investigate and compare different ranking mechanisms for ranking the co-change candidates of a target program entity. We perform such an investigation when answering our second research question (RQ 2).

B. Answering RQ 2

RQ 2: *What is the comparative scenario among different ranking mechanisms for ranking co-change candidates retrieved using evolutionary coupling?*

Motivation. Rolfsness et al. [25] previously ranked the co-change candidates using support ranking mechanism (discussed in Section III). In this section, we investigate a number of ranking mechanisms including support ranking and try to make a comparison among these mechanisms to understand which mechanism performs the best. We perform our investigation in the following way.

Investigation Procedure. As we did in RQ 1, we examine the commit operations from the beginning one. We consider each method that changed in each commit operation as a target method and apply Tarmaq algorithm for retrieving the co-change candidates of the target method from the previous commit operations. Let us again assume that the set of co-change candidates suggested by the Tarmaq algorithm for a target method is SCC. The set of methods that actually co-changed with the target method is ACC. We order the suggested co-change candidates (elements in SCC) using different ranking mechanisms and determine the average precisions.

Calculating the average precision (AP) for a particular ranking mechanism. After ordering the elements in SCC using a particular ranking mechanism, RM, we calculate the average precision in the way which was followed by Rolfsnes et al. [25]. Let us assume that the number of elements in SCC is n. We calculate the average precision for the ranking mechanism, RM, using the following equation.

$$AP_{RM} = \sum_{k=1}^{n} P@k \times \delta R@k \tag{5}$$

Here, AP_{RM} is the average precision that we obtain after ordering the elements in the set SCC using the ranking mechanism RM. P@k and R@k are the precision and recall at the k-th element in the ranked (ordered) list. We calculate these using the following equations.

$$P@k = NCS_k/k \tag{6}$$

$$R@k = NCS_k/|ACC| \tag{7}$$

In the above two equations, NCS_k is the number of correct suggestions in the first k elements of the ranked list. We previously mentioned that ACC is the set of actually co-changed candidates. Thus, NCS_k is the number of those elements (suggestions) that are present not only in the first k elements of the ranked list but also in the set ACC. $|ACC|$ is the number of elements in ACC. We calculate $\delta R@k$ using the following equation.

$$\delta R@k = R@k - R@(k-1) \tag{8}$$

Calculating the mean average precision (MAP) for a particular ranking mechanism. By considering each method that changed in each of the commit operations as a target method, we determine the two sets SCC and ACC, rank the members in SCC using a particular ranking mechanism RM, and determine the average precision AP_{RM} in suggesting the co-change candidates. Finally, we calculate the mathematical mean of all the AP_{RM} values from all the target methods in order to obtain the mean average precision MAP_{RM} for the ranking mechanism RM.

Comparing the ranking mechanisms on the basis of MAP. We determine the MAP (mean average precision) provided by each of the ranking mechanisms in ranking co-change candidates of the target methods during the entire period of evolution of each of the subject systems. Table IV shows the MAPs for different subject systems.

In Table IV, we see that different ranking mechanisms appear to be the best for different subject systems. For example, for the subject systems: Ctags, Camellia, and DNSJava, recency ranking exhibits the highest MAP. For the remaining systems except jEdit, file-proximity ranking provides the highest MAP. For jEdit, the highest MAP is obtained from similarity ranking. It is noticeable that although support ranking was being used by the existing studies and techniques, this ranking mechanism does not appear to be the best one for any subject system. Confidence ranking mechanism appears to be the worst one for all subject systems.

Statistical Significance Tests. We wanted to know if any ranking mechanism appears to be significantly better than all the other mechanisms. We perform Wilcoxon Signed Rank (WSR) test [28], [29] for this purpose. WSR test is suitable for applying on paired data samples. This test is non-parametric, and thus, the samples do not need to be normally distributed [28]. This test can be applied to both small and large data sets [28]. We apply this test considering a significance level of 5%.

As file proximity ranking gives the best MAP for the highest number of subject systems (6 out of 10 systems), we wanted to realize if this ranking mechanism is significantly better than each of the other three mechanisms. According to our test results, the MAPs provided by file-proximity ranking are significantly greater than the MAPs from the support ranking and confidence ranking mechanisms with a p-value of 0.01 for the two-tailed test case. However, file-proximity ranking does not give significantly different MAPs than the other two ranking techniques: recency ranking and similarity ranking. We also find that recency ranking is significantly better than support and confidence ranking mechanisms with a p-value of 0.04. However, the difference between recency and similarity ranking mechanisms is not significant. Finally, we did not find any particular ranking mechanism which is significantly better than all other ranking mechanisms.

Answer to RQ 2: According to our analysis on all five ranking mechanisms, we did not find any particular ranking mechanism that performs significantly better than all the other mechanisms. Moreover, different ranking mechanisms can appear to be the most promising one at different times during evolution.

Such a finding implies that sticking to a particular ranking mechanism during the entire period of evolution of a software system is not a good idea. It inspires us to investigate whether it is possible to dynamically select an appropriate ranking mechanism for the current target entity on the basis of the past occurrences of co-change. We perform such an investigation when answering RQ 3.

C. Answering RQ 3

RQ 3: *Can we select a suitable ranking mechanism on the basis of the past history for ranking co-change candidates?*

Rationale. From our previous research question we realize that different ranking mechanisms might be the most suitable one for ranking the co-change candidates of different target methods at different points of evolution of a subject system. In this subsection we investigate whether a suitable ranking mechanism can be selected on the basis of the past history of ranking co-change candidates. We perform our investigation in the following way for answering RQ 3.

Selecting the best ranking mechanism from the history for ranking the co-change candidates for a target method. Our idea of selecting the best ranking mechanism from the past ranking history is as follows. Let us assume that we are examining a commit operation. A number of methods were changed together in the commit. We consider one method, for example m, as the target method. The other methods are the actually co-changed candidates of the target method, m. We consider these other methods as the set, ACC_m. In this situation, we apply Tarmaq on the past evolutionary history of the software system and predict co-change candidates for the target method. We consider these co-change candidates as a set, SCC_m (suggested co-change candidates). From the intersection of the two sets, SCC_m and ACC_m, we get the true positives. The true positives for the target method constitute a set named TP_m. Let us also assume that the target method that we analyzed just before analyzing m is $m_{previous}$ and the target method that we will analyze just after analyzing m is m_{next}. We now perform the following two steps for the target method, m:

Step 1 (Ranking the co-change candidates of m using the previously determined best ranking mechanism): Let us assume that the best ranking mechanism in ranking the co-change candidates of $m_{previous}$ is BRM (best ranking mechanism). The way of obtaining BRM will be explained in **Step 2**. We rank the suggested co-change candidates in the set, SCC_m, using BRM. We then determine the average precision considering this BRM using Eq. 5 with the help from the sets: ACC_m and TP_m. This average precision is considered for calculating the mean average precision.

Step 2 (Analyzing the best ranking mechanism for ranking the co-change candidates of the target method m): The best ranking mechanism (BRM) that we selected in **Step 1** for ranking the members in SCC_m (i.e., the co-change candidates of m) might not actually be the best one to rank these members. BRM was the most appropriate one for ranking the co-change candidates of the previous target method $m_{previous}$. For determining which mechanism will be the best one for ranking the co-change candidates of m, we apply each of the five ranking mechanisms (discussed in Section III) on the set SCC_m and determine the average precision by analyzing the two sets ACC_m and TP_m. The ranking mechanism that will provide the highest average precision will be selected as the best ranking mechanism (BRM) for ranking the co-change candidates of the next target method m_{next} (i.e., for ranking the members in $SCC_{m_{next}}$). We should note that different ranking mechanisms may provide the same average precision. Thus, the best average precision might be obtained from more than one mechanism. In this case, we prioritize the mechanisms in the following order: (1) File-proximity Ranking, (2) Recency Ranking, (3) Similarity Ranking, (4) Support Ranking, and (5) Confidence Ranking. We consider this ordering on the basis of the number of subject systems where the ranking mechanisms appear to be the best ones. From Table IV we see that File-proximity ranking appears to be the best one for six subject systems. Recency ranking is the best one for three subject systems: Ctags, Camellia, and DNSJava. For the remaining system, jEdit, Similarity ranking provides the best MAP. Confidence ranking always appears to be the worst one. Thus, we consider this mechanism as the last one in the ordering.

Fig. 2 shows the two steps described above while ranking co-change candidates for a target method. Figure caption describes the details about the interactions between the steps.

Comparing HistoRank with the other ranking mechanisms. Fig. 3 shows the MAPs provided by the different ranking mechanisms for each of our subject systems. We see that for each of the subject systems, HistoRank provides

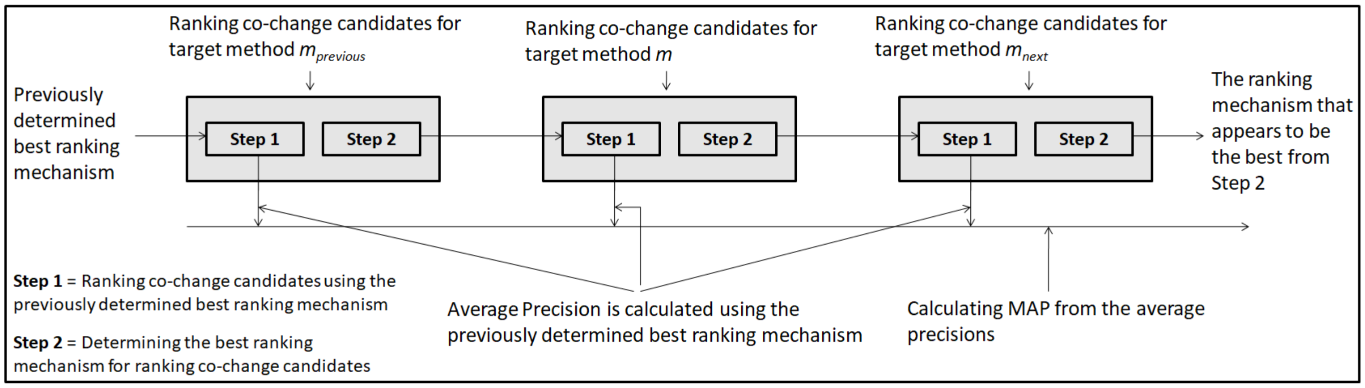
Ranking co-change candidates for target method $m_{previous}$
Ranking co-change candidates for target method m
Ranking co-change candidates for target method m_{next}
Previously determined best ranking mechanism
Step 1
Step 2
The ranking mechanism that appears to be the best from Step 2
Step 1 = Ranking co-change candidates using the previously determined best ranking mechanism
Step 2 = Determining the best ranking mechanism for ranking co-change candidates
Average Precision is calculated using the previously determined best ranking mechanism
Calculating MAP from the average precisions

Fig. 2. **Ranking co-change candidates on the basis of the previously determined best ranking mechanism**. This figure shows the two steps (Step 1, Step 2) while ranking the co-change candidates for a target method. Let us look at the big box in the middle that demonstrates ranking co-change candidates for the target method m. In Step 1 residing in this box, we get the ranking mechanism that appeared to be the best from the Step 2 while ranking the co-change candidates of the previous target method $m_{previous}$. This best ranking mechanism is applied on the co-change candidates of m for ranking those. After ranking, we determine the average precision. This average precision is considered for calculating the mean average precision (MAP). In Step 2 for the target method m, we analyze which of our four ranking mechanisms is really the best one for ranking the co-change candidates of m. This best ranking mechanism is considered in the Step 1 of the next target method m_{next} and so on.

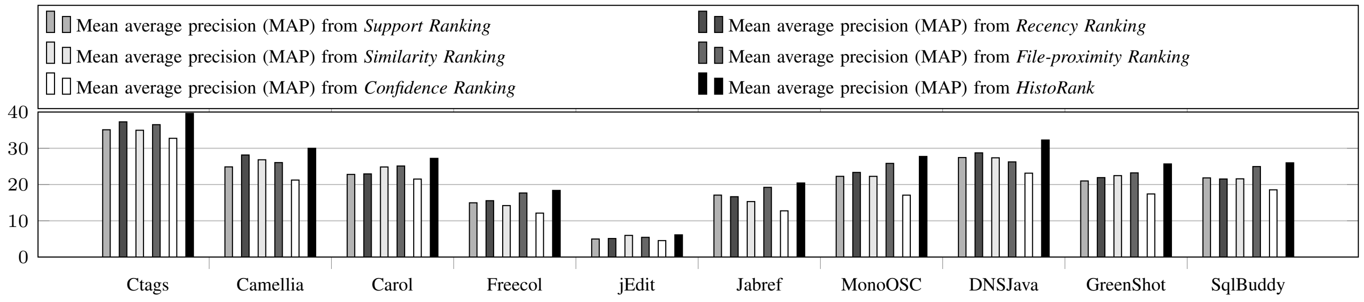

Fig. 3. Mean average precision (MAP) provided by different ranking mechanisms

the best MAP among all the ranking techniques. We also wanted to analyze whether HistoRank provides a significantly better MAP compared to the other mechanisms. We conducted Wilcoxon Signed Rank tests [28], [29] for this purpose as we did while answering RQ 2. From our test result we realize that HistoRank provides significantly higher MAPs than each of the other ranking mechanisms. The p-value for each of the test cases is **0.00512** which is smaller than 0.05. We find that the effect size [34] for each of the test cases is **0.627** which is larger than the medium effect size of 0.5 [30].

Answer to RQ 3: From our investigation and analysis we realize that it is possible to select a suitable ranking mechanism from the past ranking history for ranking the co-change candidates of a target method. Our proposed ranking approach, HistoRank, that works by analyzing the past ranking history performs the best among all the ranking mechanisms.

In the following section, we investigate different variants of HistoRank and make a comparison among those.

VI. Investigating the Variants of HistoRank

In our previous research (RQ 3), we introduced HistoRank which analyzes only the ranking in the immediate previous occurrence of co-change. In this section, we investigate two variants of the original HistoRank approach and answer the fourth research question (RQ 4) through our analysis.

RQ 4: *Which variant of history based ranking performs the best in ranking co-change candidates?*

A. *The first variant of HistoRank*

This variant also works in two steps as of the original HistoRank approach.

Step 1. In this step, for ranking the co-change candidates of the current target entity, we analyze the ranking history in all previous occurrences of co-change. We record which ranking mechanism could be the best one in which occurrence. We finally select the ranking mechanism that appeared to be the best in most of the previous occurrences of co-changes. We apply this selected ranking mechanism for ranking the co-change candidates of the current target program entity. The average precision obtained by applying this selected mechanism is used for calculating the MAP.

Step 2. Again, the ranking mechanism that we selected in **Step 1** by analyzing the ranking occurrences in the previous history might actually not appear to be the best one for the current target entity. We apply all our ranking mechanisms for ranking the co-change candidates of the current target entity, determine the best mechanism by analyzing its actually co-

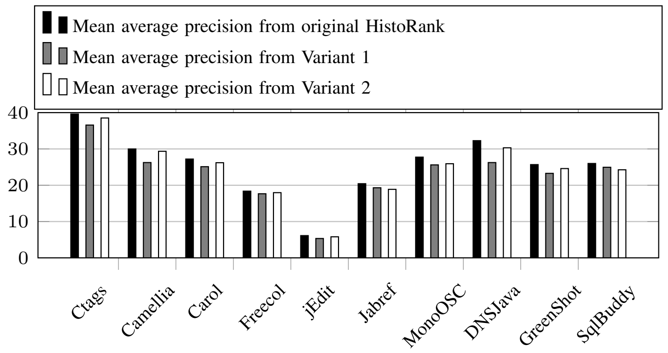

Fig. 4. Percentage of suggested co-change candidates that are true positives

changed candidates, and record this best mechanism for using in the calculation for the next target entity.

B. The second variant of HistoRank

In this variant we maintain a separate history for each of the target entities. Ranking co-change candidates for a target program entity consists of the following two steps.

Step 1. In this step, we first determine if we previously ranked co-change candidates of the same target entity. If we did this, the best ranking mechanism from the latest occurrence of ranking is already recorded. We use this recorded mechanism for ranking the recent set of co-change candidates of the target entity. The average precision that we obtain by using this recorded mechanism is considered for calculating MAP. However, if the same target entity was not previously seen, we select the ranking mechanism that was decided to be the best in the immediate previous occurrence of ranking. The average precision obtained after applying this previously decided best mechanism is considered for calculating MAP.

Step 2. In this step, we apply all five ranking mechanisms (Section III) on the co-change candidates of the target entity, determine the best mechanism by analyzing the actually co-changed candidates of the target, and finally record this best mechanism against the target entity so that if we need to deal with this target again in the future, we can use this recorded ranking mechanism.

C. Comparing the original HistoRank with its variants

Fig. 4 shows the comparison of the original HistoRank approach with its variants. According to the graph, the original HistoRank has the best MAP for each of our subject systems. Between the two variants, the second variant that maintains a separate history for each target entity is mostly better than the first variant. Moreover, the performance of the second variant is often very near to the original HistoRank approach. The comparison scenario among the three approaches makes us realize that emphasizing the best ranking mechanism in the latest occurrence of co-change in the history provides us the best MAP for each of the subject systems.

We also wanted to determine whether the original HistoRank is significantly better than its two variants. We again perform Wilcoxon Signed Rank tests [28], [29] for this purpose considering a significance level of 5%. We compare the MAPs provided by the original HistoRank with the MAPs provided by each of the two variants. From our tests we realize that the original HistoRank provides significantly better MAPs than each variant with a p-value of 0.005 and a Cohen's d effect size [34] of 0.627.

Answer to RQ 4. According to our analysis, the original HistoRank approach that emphasizes the ranking in the latest occurrence of co-change in the history performs the best in ranking co-change candidates.

VII. Discussion

While the existing studies dealing with association rules for detecting co-change candidates have always ranked the candidates using the support and confidence measures, our analysis for answering RQ 2 shows that ranking on the basis of these measures is often not desirable. The other ranking mechanisms such as recency ranking, similarity ranking, and file-proximity ranking appear to be the most promising ones at different points of evolution of our subject systems. Our answer to RQ 2 makes us realize that sticking to a particular ranking mechanism during the entire evolution might not be a good idea. Our investigation regarding RQ 3 analyzes the possibility of dynamically selecting the most appropriate ranking mechanism from the past ranking history of co-change candidates. As a result of this analysis, we propose a ranking approach, HistoRank, that is capable of dynamically selecting the most promising ranking mechanism by analyzing the history. In particular, HistoRank selects which of the five mechanisms discussed in Section III can be the most suitable one for ranking the co-change candidates of a target program entity. Our analysis in RQ 4 on different variants of HistoRank implies that the approach that emphasizes the most recent occurrence of co-change in the past performs the best. Our implementation of HistoRank is available online [32].

VIII. Threats to Validity

We did not investigate enough subject systems to generalize our findings regarding ranking co-change candidates obtained using evolutionary coupling. However, to ensure our findings were worthwhile, we countered this threat by selecting a diverse set of candidate systems, in terms of application domains, sizes, number of revisions, and by covering three different programming languages.

The revision history lengths of our investigated subject systems might not be enough to generalize our findings. However, according to Table II, the revision histories of our subject systems are of different lengths from small to large. Thus, our experiment was not biased by the revision history lengths of our subject systems.

IX. Related Work

Evolutionary coupling is a well investigated phenomenon in the realm of software engineering research and practice. A great many studies have been done on the detection [1],

[7], [8], [13], [17], [18], [25], analysis [6], [11], [36], and usage [3], [9], [10], [14]–[16], [19], [21], [22] of evolutionary coupling. One of the major purposes of using evolutionary coupling is suggesting co-change candidates [5], [9], [13], [25] whenever a programmer makes changes to the code-base. Our research in this paper focuses on ranking the co-change candidates suggested by evolutionary coupling.

Mondal et al. [16] previously used recency ranking for ranking co-change candidates of code clones. The co-change candidates of a target clone fragment are simply the other fragments in the same clone group detected by a clone detector. Our investigation also involves recency ranking. However, our study is different because we apply recency ranking for ranking co-change candidates for methods. We use association rules for predicting co-change candidates for a target method. Our experiment shows that recency ranking performs significantly better than support and confidence ranking when ranking co-change candidates for methods. We have also experimented different other ranking mechanisms and finally propose a history based ranking approach, HistoRank, that significantly outperforms each of the ranking mechanisms.

Zimmerman et al. [27] implemented a tool called Rose that can detect evolutionary coupling among different types of program entities through mining version histories under CVS. According to their investigation, evolutionary coupling can detect coupling links which cannot be detected by program analysis. Our study is different from their study because we investigate ranking of co-change candidates obtained through analyzing evolutionary coupling. Zimmerman et al. [27] did not perform such an investigation.

Rolfsness et al. [25] introduced an algorithm called Tarmaq that analyzes evolutionary coupling for predicting co-change candidates for files. They showed that Tarmaq outperforms Rose and SVD in detecting impact sets. They ranked the co-change candidates (i.e., the items in the impact set) using support and confidence measures. In our study, we use the Tarmaq algorithm for predicting co-change candidates for methods. We rank the co-change candidates for methods using five different ranking mechanisms (Section III). We finally propose a ranking approach, HistoRank, that dynamically selects the best ranking mechanism on the basis of the ranking occurrences in the past evolutionary history.

Pugh et al. [36] investigated if it is possible to achieve an accuracy level like Tarmaq using a smaller number of commit transactions than Tarmaq. They showed that a dynamically selected smaller set of commit transactions can be enough instead of considering the whole set of transactions from the entire past history. However, our research goal is different. We investigate which ranking mechanism can be the most suitable one for ranking the co-change candidates retrieved using evolutionary coupling.

Islam et al. [13] investigated whether it is possible to improve the detection accuracy of evolutionary coupling by applying transitivity on regular association rules. Kagdi et al. [9] investigated if it is possible to provide better co-change suggestions by combining evolutionary coupling and conceptual coupling. However, these studies did not investigate ranking co-change suggestions.

A number of other studies have also investigated how to improve the detection accuracy of evolutionary coupling using various measures. For example, Alali et al. [1] investigated two measures: pattern age and pattern distance for this purpose. Mondal et al. proposed two measures: significance and change correspondence in two preliminary studies [17], [18]. Jafaar et al. [6] investigated macro co-change and dephase macro co-change for improving the detection accuracy. Canfora et al. [7] introduced Granger causality test for the same purpose. Bantelay et al. [5] combined interactions with commit operations for improved detection of evolutionary coupling. While all these studies proposed different measures and used different techniques for improved detection of evolutionary coupling, none of these studies investigated ranking of the co-change suggestions. We investigate ranking of the co-change suggestions in our study.

All the existing studies that deal with association rules for detecting co-change candidates for program entities have ranked the candidates using the support and confidence measures. Our study is different from all these existing studies because we investigate which of the different ranking mechanisms can be the most appropriate one for ranking co-change candidates. We analyze five different ranking mechanisms and finally propose a ranking approach called HistoRank that significantly outperforms all those five mechanisms. We believe that HistoRank will be an important addition to the existing knowledge of evolutionary coupling.

X. Conclusion

In our study, we investigate five different ranking mechanisms for ranking the co-change candidates obtained through applying evolutionary coupling. We apply these ranking mechanisms on thousands of revisions of ten subject systems and realize that different mechanisms can appear to be the most suitable one at different points of evolution of a software system. We finally propose a history-based ranking approach, HistoRank, that ranks the co-change candidates by analyzing the past occurrences of ranking. We find that HistoRank performs the best for each of our subject systems. We investigate two variants of the original HistoRank approach and find that the original one that emphasizes the most recent occurrence of co-change in the past evolution performs the best. We believe that our proposed HistoRank ranking approach can be important for the programmers in identifying the most likely co-change candidates during programming. The implementation and data from our research are available online [32].

Acknowledgement

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), and by two Canada First Research Excellence Fund (CFREF) grants coordinated by the Global Institute for Food Security (GIFS) and the Global Institute for Water Security (GIWS).

References

[1] A. Alali, B. Bartman, C. D. Newman, J. I. Maletic, "A Preliminary Investigation of Using Age and Distance Measures in the Detection of Evolutionary Couplings", in Proc. *MSR*, 2013, pp. 169 – 172.

[2] A. Lozano, M. Wermelinger, "Assessing the effect of clones on changeability", Proc. *ICSM*, 2008, pp. 227 – 236.

[3] C. Tantithamthavorn, A. Ihara, K. Matsumoto,"Using Co-Change Histories to Improve Bug Localization Performance", Proc. *ACIS*, 2013, pp. 543 – 548.

[4] CTAGS: http://ctags.sourceforge.net/.

[5] F. Bantelay, M. B. Zanjani, H. Kagdi, "Comparing and Combining Evolutionary Couplings from Interactions and Commits", in Proc. *WCRE*, 2013, pp. 311 – 320.

[6] F. Jaafar, Y. Gueheneuc, S. Hamel, G. Antoniol,"An Exploratory Study of Macro Co-changes", Proc. *WCRE*, 2011, pp. 325 – 334. .

[7] G. Canfora, M. Ceccarelli, L. Cerulo, M. D. Penta,"Using Multivariate Time Series and Association Rules to Detect Logical Change Coupling: an Empirical Study", Proc. *ICSM*, 2010, pp. 1 – 10.

[8] H. Gall, M. Jazayeri, J. Krajewski,"CVS Release History Data for Detecting Logical Couplings", Proc. *IWPSE*, 2003, pp. 13 – 23.

[9] H. Kagdi, M. Gethers, D. Poshyvanyk, M. L. Collard,"Blending Conceptual and Evolutionary Couplings to Support Change Impact Analysis in Source Code", Proc. *WCRE*, 2010, pp. 119 – 128.

[10] J. Itkonen, M. Hillebrand, V. Lappalainen, "Application of Relation Analysis to a Small Java Software", Proc. *CSMR*, 2004, pp.233-239.

[11] L. Moonen, T. Rolfsnes, D. Binkley, and S. Di Alesio, "What are the effects of history length and age on mining software change impact?", Empirical Software Engineering (EMSE), no. https://doi.org/10.1007/s10664-017-9588-z, pp. 1–36, 2018.

[12] L. R. Dice, "Measures of the Amount of Ecologic Association Between Species", *Ecology*, 1945, 26(3): 297 – 302.

[13] M. A. Islam, M. M. Islam, M. Mondal, B. Roy, C. K. Roy, K. A. Schneider, "Detecting Evolutionary Coupling Using Transitive Association Rules", Proc. *SCAM*, 2018, pp. 113 – 122.

[14] M. D'Ambros, M. Lanza,"Reverse Engineering with Logical Coupling", Proc. *WCRE*, 2006, pp. 189 –198.

[15] M. Mondal, C. K. Roy, K. A. Schneider, "Automatic Ranking of Clones for Refactoring through Mining Association Rules", Proc. *CSMR-WCRE*, 2014, pp. 114 – 123.

[16] M. Mondal, C. K. Roy, K. A. Schneider, "Prediction and ranking of co-change candidates for clones", Proc. *MSR*, 2014, pp. 32 – 41.

[29] Wilcoxon Signed Rank Test. https://www.socscistatistics.com/tests/signedranks/Default2.aspx.

[17] M. Mondal, C. K. Roy, K. A. Schneider,"Improving the detection accuracy of evolutionary coupling by measuring change correspondence", Proc. *WCRE-CSMR*, 2014, pp. 358 –362.

[18] M. Mondal, C. K. Roy, K. A. Schneider,"Improving the Detection Accuracy of Evolutionary Coupling", Proc. *ICPC*, 2013, pp. 223 – 226.

[19] N. Hanakawa,"Visualization for software evolution based on logical coupling and module coupling", Proc. *APSEC*, 2007, pp. 214 – 221.

[20] R. Agrawal, T. Imielinski, A. Swami, "Mining Association Rules between Sets of Items in Large Databases", *ACM SIGMOD*, 1993, 22(2):207 – 216.

[21] S. N. Ahsan, F. Wotawa,"Fault Prediction Capability of Program File's Logical-Coupling Metrics", Proc. *IWSM-MENSURA*, 2011, pp.257 - 262.

[22] S. Wenzel, H. Hutter, U. Kelter,"Tracing Model Elements", Proc. *ICSM*, 2007, pp. 104 – 113.

[23] Sørenson–Dice coefficient: https://en.wikipedia.org/wiki/S\%C3\%B8rensen\%E2\%80\%93Dice_coefficient.

[24] Strike A Match: http://www.catalysoft.com/articles/strikeamatch.html.

[25] T. Rolfsnes, S. D. Alesio, R. Behjati, L. Moonen and D. W. Binkley, "Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis", Proc. *SANER*, 2016, pp. 201 – 212.

[26] T. Sørensen, "A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons", *Kongelige Danske Videnskabernes Selskab*, 1948, 5(4): 1 – 34.

[27] T. Zimmermann, P. Weissgerber, S. Diehl, A. Zeller,"Mining Version Histories to Guide Software Changes", Proc. *ICSE*, 2005, pp. 563 – 572.

[28] Wilcoxon Signed Rank Test. https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test.

[30] Effect Size and Interpretation: https://en.wikipedia.org/wiki/Effect_size.

[31] The evolution radar: visualizing integrated logical coupling information.

[32] Implementation and Data: https://drive.google.com/open?id=19SmdhyfiEcK2CYNHwhs8bU2fsPX70KCE.

[33] Online SVN repository: http://sourceforge.net/.

[34] Wilcoxon Signed Rank Test and Cohen's d effect size calculator: https://www.ai-therapy.com/psychology-statistics/effect-size-calculator.

[35] A. Farhanah, R. Mohamad, and N. Ibrahim. Comparative evaluation of string metrics for context ontology database. *Journal of Telecommunication, Electronic and Computer Engineering*, 9(3-3):7 – 11, 2017.

[36] S. Pugh, D. Binkley, and L. Moonen. The case for adaptive change recommendation. In *SCAM*, pages 129 – 138, 2018.