The Case for Adaptive Change Recommendation

Sydney Pugh*, David Binkley*, Leon Moonen§

*Loyola University Maryland, 4501 N. Charles St., Baltimore, MD 21210-2699, USA
§Simula Research Laboratory, Oslo, Norway

Abstract—As the complexity of software systems grows, it becomes increasingly difficult for developers to be aware of all the dependencies that exist between artifacts (e.g., files or methods) of a system. Change impact analysis helps to overcome this problem, as it recommends to a developer relevant source-code artifacts related to her changes. Association rule mining has shown promise in determining change impact by uncovering relevant patterns in the system’s change history.

State-of-the-art change impact mining algorithms typically make use of a change history of tens of thousands of transactions. For efficiency, targeted association rule mining is used because it focuses on only those transactions relevant to answering a particular query. However, even targeted algorithms must consider the complete set of relevant transactions in the history.

This paper presents ATARI, a new adaptive approach to targeted association rule mining that considers a dynamic selection of relevant transactions. It can be viewed as further constrained version of targeted association rule mining, in which as few as a single transaction might be considered when determining change impact. Our investigation of adaptive change impact mining empirically studies seven algorithm variants. We show that adaptive algorithms are viable, can be just as applicable as the start-of-the-art complete-history algorithms, and even outperform them for certain queries. However, more important than the direct comparison, our investigation lays important groundwork to support the application of adaptive techniques to problems such as, impact analysis at the GitHub-scale.

I. INTRODUCTION

When software systems evolve, the interactions in the source code grow in number and complexity. As a result, it becomes increasingly challenging for developers to predict the overall effect of making a change to the system. Aimed at identifying software artifacts (e.g., files, methods, classes) affected by a given change, Change Impact Analysis [1] has been proposed as a solution to this problem. Traditionally, techniques for change impact analysis have been based on static or dynamic analysis, which identifies dependencies, for example, methods calling or called by a changed method [2–4]. However, static and dynamic analysis are generally language-specific, making them hard to apply to modern heterogeneous software systems [5]. In addition, dynamic analysis can involve considerable overhead (e.g., from code instrumentation), while static analysis tends to over-approximate the impact of a change [6].

To address these challenges, alternative techniques have been proposed that identify dependencies through evolutionary coupling [7–10]. In essence, these techniques exploit a developer’s inherent knowledge of dependencies in the system, which manifest themselves through, for example, commits and their comments [11], bug reports and fixes [12], and IDE activity [13].

This paper uses historical co-change between artifacts as the basis for uncovering evolutionary coupling. It does this using a variation on targeted association rule mining [14]. We refer to the process of using mined evolutionary couplings for change impact analysis as change impact mining.

Existing algorithms for change impact mining [10, 15–18] consider the complete set of transactions in the change history. Recent work with TARMQ on the impact of history length on analysis quality [19] shows that short histories limit the algorithm’s ability to give answers, but when it can, the average quality of those answers is high. This suggests the potential value of adaptively deciding how much history to use.

In this paper, we propose a new approach, ATARI (Adaptive Targeted Association Rule mining), and empirically investigate several variants of adaptive change impact mining algorithms that vary in how they determine the amount of history to consider. Our hypothesis is that the reduced number of transactions considered by adaptive targeted association rule mining will improve on targeted association rule mining [14] akin to how targeted association rule mining improved on association rule mining [20].

Our motivation for studying adaptive techniques is two-fold: first, we seek to better understand the interplay between the transaction history and the mining result. In its classical applications (e.g., to shopping cart data), association rule mining typically requires a large amount of data. To date, all existing applications of association rule mining to change recommendation have blindly assumed that the same is true in the software context. However, preliminary experiments suggest that software is somehow fundamentally different and thus might benefit from software-specific algorithms.

Second, we want to create algorithms that are sufficiently efficient for integration of change impact mining with online services such as GitHub. In general, making a recommendation is quick, even when using the entire relevant history. However, extracting the complete history takes considerably longer. For an online service like GitHub, the space and time needed to keep up-to-date extracted histories for all projects becomes an issue, making on-the-fly adaptive analysis preferable.

This paper makes the following contributions:

• it introduces the concept of adaptive targeted association rule mining,
• it proposes several adaptive algorithms for change impact mining,
• it studies the implementation of these algorithms in the prototype tool, ATARI, and finally

Accepted for publication in the 18th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2018) in Madrid, Spain. This is a preprint to ensure timely dissemination of scholarly work. Copyright and all rights therein are retained by authors or by other copyright holders. This work may not be reposted, republished, duplicated, or distributed to any entity, individuals or individual without the explicit permission of the copyright holder. The definitive version will be, or has been, published by IEEE in the proceedings of SCAM 2018.
• it compares the new algorithms to each other and to the state-of-the-art, TARMAQ [10].

The remainder of the paper provides background on association rule mining and its application for change impact mining in Section II and III. The newly devised adaptive techniques are presented in Section IV, followed by their study in Sections V through VII. The paper concludes with a discussion of related work in Section VIII and a summary in Section IX.

II. ASSOCIATION RULE MINING

This section provides the basic definitions used to frame the problem of change impact analysis using association rule mining, an unsupervised learning technique used to discover relations between the artifacts of a dataset [20]. Association rules are implications of the form \( A \rightarrow B \), where \( A \) is referred to as the antecedent, \( B \) as the consequent, and \( A \) and \( B \) are disjoint sets. For example, consider the classic application of analyzing shopping-cart data: if multiple transactions include bread and butter then a potential association rule is \( \text{bread} \rightarrow \text{butter} \), which can be read as “if you buy bread, then you are likely to buy butter.”

In our application, the input to the algorithm is a history of transactions, denoted \( T \), where each transaction originates with a commit from a versioning system. More specifically, a transaction \( T \in T \) is the set of artifacts (files or functions) that were either changed or added while addressing a given bug or feature addition, hence creating a logical dependence between the artifacts [21]. In contrast to applications outside of software engineering, where the history is treated as a set, we treat the history as an age ordered list and define two transactions as adjacent if they occur one right after the other in the history.

Targeted association rule mining [14] focuses the generation of rules by applying a constraint, which often dramatically improves rule generation time. In change impact mining, rule generation is constrained by a change set, also known as a query. An example change set would be the set of artifacts modified since the last commit. In this case, only transactions with at least one modified artifact would need to be considered during rule generation.

Definition 1 (Relevant Transaction): Given a query \( q \), a transaction \( t \in T \) is relevant iff \( t \cap q \neq \emptyset \) and \( t - q \neq \emptyset \). (the second requirement ensures that the transaction contains at least one artifact to recommend).

It is possible that a mining algorithm is unable to determine the impact of a change (e.g., given a query of artifacts not found in the history). When it is possible, we define the algorithm to be applicable to the query. Otherwise, we say that the algorithm is not applicable to the query. Other things being equal, higher applicability is preferred.

When a query is applicable, the impacted artifacts are found in the consequents of mined rules. In other words, the list of artifacts that historically changed alongside elements of the query. This list can be ranked based on support and confidence [20]. The support of a rule is the percentage of transactions in the history containing both its antecedent and its consequent. Intuitively, high support suggests that a rule is more likely to hold because there is more historical evidence for it. On the other hand, the confidence of a rule is the number of historical transactions containing both the antecedent and the consequent, divided by the number of transactions containing only the antecedent. Intuitively, the higher the confidence, the higher the chance that when items in the antecedent change, the items in the consequent also change. Rules are ranked based on support, breaking ties using confidence.

Finally, transactions do not record the order of the individual changes involved. Hence, in the evaluation we empirically assess the quality of an impact mining algorithm by repeatedly selecting a transaction \( t \) from its change history and randomly splitting the transaction into a non-empty query, \( q \), and a non-empty expected-outcome, \( E_q \) that will serve as the ground truth. This splitting approach, which yields a uniform distribution of query and expected-outcome sizes, is standard in the evaluation of change impact mining techniques [10, 15–18]. The query \( q \) is then used to mine an impact set \( I_q \) using only transactions from the history that are older than \( t \). This mimics a developer in the process of making the change covered by \( t \), but forgetting one or more artifacts (those of \( E_q \)). The quality of the mined impact \( I_q \) is assessed using its Average Precision (AP) [22], while the quality of an algorithm is assessed by its mean AP (MAP) over a collection of queries.

The experiments make use of two different MAP computations: overall MAP and MAP when applicable. The difference lies in the treatment of queries for which an algorithm is not applicable. While it is possible to assign such queries an AP of zero, doing so is harsh because the algorithm can correctly inform the user that it is not applicable. From an engineer’s perspective, being given a wrong list (where AP is truly zero) is far worse than being told that no recommendation is possible. The other alternative is to ignore such queries, which is optimistic especially when comparing two algorithms applied to a particularly challenging query where one is applicable and the other is not. In this case the non-applicable algorithm should likely see some penalty. Both possibilities are considered. Overall MAP is computed by assigning non-applicable queries the AP value zero, while MAP when applicable is computed by ignoring such queries.

III. EXISTING TECHNIQUES

Only a handful of targeted association rule mining algorithms have been considered in the context of change impact analysis. The oldest two, ROSE [15] and the application of FP-TREE to change recommendation [16], were independently developed around the same time. Both ROSE and FP-TREE only uncover artifacts that changed in the history together with all entities of the query. This strict requirement leaves these algorithms unable to make a recommendation more often than not [10].

At the other end of the spectrum the Co-CHANGE algorithm [17] uncovers artifacts that co-changed with any element of the query. This lenient requirement yields more answers, which are, however, potentially noisy, as they can have little relation to the full query.

Finally, the more recent TARMAQ algorithm [10] attempts to balance these two. It uncovers the artifacts that have co-
IV. ADAPTIVE TECHNIQUES

This section introduces the seven families of adaptive techniques considered in our study. We refer to them as families because the latter four are parameterized and give rise to multiple algorithms (the first three are unparameterized, i.e., families of one). To illustrate each family, Figure 1 shows the treatment of the following history: \( T = t_1, T_2, T_3, T_4, t_5, T_6, t_7, T_8, T_9, T_{10}, T_{11} \), where uppercase denotes a relevant transaction, lowercase a non-relevant transaction, and \( t_1 \) is the most recent transaction.

Change impact mining involves four steps: (1) select relevant transactions from which to build rules, (2) generate rules from the selection by matching against the query, (3) rank the rules, and (4) provide a recommendation based on the highest ranked rules. The adaptive techniques presented here all affect the first step: based on the query, they dynamically select relevant transactions from which to generate rules. If this step produces an empty list, then no recommendation is possible. Steps (2) through (4) are unchanged from the TARMQ algorithm [10].

The simplest adaptive algorithm, first-applicable, selects the first relevant transaction. Like all the adaptive algorithms it begins this search from the most recent transaction. In the example this is transaction \( T_2 \). There are two motivations for considering the first-applicable transaction: first, there is evidence from previous analysis [19] that a (single) recent relevant transaction produces very good results. In addition, this algorithm provides a useful baseline as it is the simplest possible adaptive approach.

At the other end of the spectrum, the dynamic-all algorithm returns all relevant transactions. Dynamic-all is used as a strawman to investigate the value brought by the more selective adaptive algorithms.

The third adaptive algorithm, dynamic-block, aims to exploit the observation that adjacent relevant transactions (i.e., those immediately following a relevant transaction) are likely made by a single developer who frequently commits changes while working on a given bug or enhancement. Dynamic-block starts with the first relevant transaction and then includes all subsequent adjacent relevant transactions. In the example, the transactions selected are \( T_2, T_3, \) and \( T_4 \).

The next family, dynamic-P\( n \), is motivated by the observation that dynamic-block might perform poorly if a developer does not commit frequently or if there are a large number of developers working in parallel. In both of these situations the block size is likely small because of the increased likelihood of interjected unrelated commits in the sequence of relevant commits. Dynamic-P\( n \) provides a more tolerant approach. It starts with the first relevant transaction and includes subsequent transactions until the percentage of relevant transactions considered falls below \( n \% \). For example, dynamic-P\( 70 \) yields \( T_2, T_3, T_4, T_5 \), where the percentage of relevant transactions at each step is 100% (1/1), 100% (2/2), 100% (3/3), 75% (3/4), 80% (4/5), and finally 66% (4/6). We consider the three family members dynamic-P\( 20 \), dynamic-P\( 50 \), and dynamic-P\( 80 \).

The fifth family, dynamic\( _{1,n} \), is another attempt at greater tolerance. It starts with the first relevant transaction and includes the next \( n \) relevant transactions. If there are not \( n \) relevant transactions in the history, then an empty list is returned, which means that no recommendation is possible. It is important to note here that no partial lists (with less than \( n \) transactions) are considered. This choice enables the analysis to more accurately compare dynamic\( _{1,2} \) and dynamic\( _{1,3} \) for example, because dynamic\( _{1,3} \) does not also include elements from dynamic\( _{1,2} \).

Using the example history, dynamic\( _{1,5} \) yields \( T_2, T_3, T_4, T_5, T_6 \). Our study considers the eight dynamic\( _{1,n} \) family members for \( n \in \{1, 2, 3, 4, 5, 10, 100, 1000\} \).

The next family, dynamic\( _{1,n}\text{ adjacent} \) is a variation on dynamic\( _{1,n} \). It starts with the first relevant transaction and includes the next \( n \) adjacent relevant transactions. When there are not \( n \) adjacent relevant transactions, an empty list is returned. For example, dynamic\( _{1,2}\text{ adjacent} \) yields \( T_2, T_3 \), while dynamic\( _{1,5}\text{ adjacent} \) yields the empty list. We consider the five dynamic\( _{1,n}\text{ adjacent} \) family members for \( n \in \{1, 2, 3, 4, 5\} \).

The last family, dynamic\( _{n}\text{ adjacent} \), aims to increase dynamic\( _{1,n}\text{ adjacent} \)’s low applicability while preserving its high average precision. Dynamic\( _{n}\text{ adjacent} \) uses the most recent \( n \) adjacent relevant transactions. It differs from dynamic\( _{1,n}\text{ adjacent} \) in that it is not anchored at the first relevant transaction. For example, dynamic\( _{5}\text{ adjacent} \) yields \( T_8, T_9, T_{10}, T_{11} \). If there are not \( n \) adjacent relevant transactions anywhere in the history, then an empty list is returned. We study the five dynamic\( _{n}\text{ adjacent} \) family members for \( n \in \{1, 2, 3, 4, 5\} \).

V. RESEARCH QUESTIONS

We aim to better understand the pros and cons of using adaptive algorithms for mining association rules through the following research questions:

- **RQ1 Viability**: Do there exist any queries for which the performance of the adaptive algorithms is comparable to that of TARMQ?
- **RQ2 Applicability**: How does the applicability of the adaptive algorithms compare to each other and to that of TARMQ? (This includes inter-family comparisons.)

<table>
<thead>
<tr>
<th>figure1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
Fig. 1. A visualization of the various adaptive algorithms
```
between the various adaptive approaches, and intra-family comparisons for different values of $n$.)

**RQ3 Accuracy:** How do the MAP values of the adaptive algorithms compare to each other and to that of TARMAC? Accuracy considers both overall MAP and MAP when applicable.

**RQ4 Adjacency’s Impact:** What impact does requiring the selected transactions to be adjacent bring in terms of both applicability and accuracy?

**RQ5 Practical Implications:** What effect do the adaptive algorithms have on the overall time and space needed to make a GitHub-scale recommendation?

### VI. Empirical Investigation

This section describes the experimental design. It first discusses the software systems studied and then describes how the queries used in the experiments were created. Finally, we discuss our prototype implementation.

#### A. Subject Systems

To assess the adaptive algorithms, we selected 19 large systems having varying characteristics, such as size and frequency of transactions, number of artifacts, and number of developers. Two of these are industrial systems, from Cisco Norway and Kongsberg Maritime (KM), respectively. Cisco is a worldwide leader in the production of networking equipment. We consider a software product line for professional video conferencing systems made by Cisco Norway. KM is a leader in the production of systems for positioning, surveying, navigation, and automation of vessels and offshore installations. We consider a software platform that is used across their systems.

The other 17 systems are well known open-source projects, and are reported in Table I along with demographics illustrating their diversity. For each system, we extracted the 50,000 most recent transactions (commits). This number of transactions covers vastly different time spans across the systems, ranging from almost 20 years in the case of HTTPD, to a little over 10 months in the case of the Linux kernel.

Finally, we consider practical fine-grained histories [23] that contain function-level granularity for source code files that srcML [24] can parse, and file-level granularity otherwise. We include a residual per file to capture all changes to code that is not part of a function (e.g., global variable declarations).

#### B. Query Generation

Conceptually, a query $q$ represents a set of artifacts that a developer changed since the last synchronization with the version control system. Recall that the key assumption behind evolutionary coupling is that artifacts that frequently change together are likely to depend on each other. This is often not true of large transactions such as mass license updates or version bumps. Fortunately, transaction sizes are heavily skewed towards smaller transactions. Unfortunately, there exist outlier transactions containing 10,000 or more artifacts. Thus, it is common practice to filter the history by removing transactions larger than a certain size [15, 16, 18, 25].

In an attempt to reflect most change impact analysis scenarios, we employ a quite liberal filtering and remove only those transactions larger than 300 artifacts. The rationale behind choosing this cutoff is that for each program at least 99% of all transactions are smaller than 300 artifacts. In most cases, the percentage is well above 99% of the available data.

Finally, to generate a set of queries to experiment with, we randomly sample 1100 recent transactions from each filtered history. Each selected transaction, $t$, is then randomly split into a non-empty query and a non-empty expected outcome. Finally, to respect the historical time-line, the history used is composed of the transactions older than $t$.

#### C. ATARI Implementation

The seven families of algorithms from Section IV were implemented in a prototype tool ATARI (Adaptive Targeted Association Rule mining). ATARI was developed in RUBY, and built as a fork of the TARMAC implementation, which was generously provided to us by its developers [10]. Because both tools use the same input- and output-formats, it was easy to compare the experimental results.

### VII. Results and Discussion

#### A. RQ1: Viability

RQ1 asks if any of the adaptive algorithms ever out perform TARMAC. This research question considers the viability of the adaptive approach. Given that TARMAC has access to ten’s of thousands of transactions, it is not unreasonable to expect that it will always outperform any of the adaptive algorithms,

1 For a normally distributed population of 50,000, a minimum of 657 samples is required to attain 99% confidence with a 5% confidence interval that the sampled transactions are representative of the population. To account for non-normality we increase the sample size using the lowest (most conservative) Asymptotic Relative Efficiency (ARE) correction coefficient, 0.637, yielding a sample size of 657/0.637 = 1032 transactions. Hence, a sample size of 1100 is more than sufficient to attain 99% confidence with a 5% confidence interval that the samples are representative of the population.

<table>
<thead>
<tr>
<th>System</th>
<th>History (years)</th>
<th>Unique files</th>
<th>Unique artifacts</th>
<th>Languages used*</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPython</td>
<td>12.05</td>
<td>7725</td>
<td>30090</td>
<td>Python, C, 16 others</td>
</tr>
<tr>
<td>GeoC</td>
<td>1.08</td>
<td>86650</td>
<td>231850</td>
<td>C++, C, JavaScript, 34 others</td>
</tr>
<tr>
<td>Git</td>
<td>11.02</td>
<td>3753</td>
<td>17716</td>
<td>C, shell script, Perl, 14 others</td>
</tr>
<tr>
<td>Hadoop</td>
<td>6.91</td>
<td>24607</td>
<td>272902</td>
<td>Java, XML, 10 others</td>
</tr>
<tr>
<td>HTTPD</td>
<td>19.78</td>
<td>10019</td>
<td>29216</td>
<td>XML, C, Forth, 19 others</td>
</tr>
<tr>
<td>IntelliJ IDEA</td>
<td>2.61</td>
<td>62692</td>
<td>343613</td>
<td>Java, Python, XML, 26 others</td>
</tr>
<tr>
<td>Liferay Portal</td>
<td>0.87</td>
<td>144792</td>
<td>767955</td>
<td>Java, XML, 12 others</td>
</tr>
<tr>
<td>Linux Kernel</td>
<td>0.77</td>
<td>26412</td>
<td>161022</td>
<td>C, 16 others</td>
</tr>
<tr>
<td>LLVM</td>
<td>4.54</td>
<td>25600</td>
<td>66604</td>
<td>C++, Assembly, C, 16 others</td>
</tr>
<tr>
<td>MediaWiki</td>
<td>11.69</td>
<td>12252</td>
<td>12267</td>
<td>PHP, JS, 11 others</td>
</tr>
<tr>
<td>MySQL</td>
<td>10.68</td>
<td>42589</td>
<td>136925</td>
<td>C++, C, JS, 24 others</td>
</tr>
<tr>
<td>PHP</td>
<td>10.82</td>
<td>21295</td>
<td>55350</td>
<td>C, PHP, XML, 24 others</td>
</tr>
<tr>
<td>Ruby on Rails</td>
<td>11.42</td>
<td>10631</td>
<td>10631</td>
<td>Ruby, 6 others</td>
</tr>
<tr>
<td>RavenDB</td>
<td>8.59</td>
<td>29245</td>
<td>139415</td>
<td>C#, JS, XML, 12 others</td>
</tr>
<tr>
<td>Subversion</td>
<td>14.03</td>
<td>6359</td>
<td>46136</td>
<td>C, Python, C++, 15 others</td>
</tr>
<tr>
<td>WebKit</td>
<td>3.33</td>
<td>281898</td>
<td>397850</td>
<td>HTML, JS, C++, 24 others</td>
</tr>
<tr>
<td>Wine</td>
<td>6.6</td>
<td>8234</td>
<td>126177</td>
<td>C, 16 others</td>
</tr>
<tr>
<td>Cisco</td>
<td>2.43</td>
<td>64974</td>
<td>251321</td>
<td>C++, C, C#, Python, others</td>
</tr>
<tr>
<td>KM</td>
<td>15.97</td>
<td>35111</td>
<td>35111</td>
<td>C++, C, XML, others</td>
</tr>
</tbody>
</table>

* languages used by open source systems from http://www.openhub.net.
especially given that most of the adaptive algorithms use orders of magnitude fewer transactions.

When comparing two algorithms $A$ and $B$, the two tie when applied to a query if neither is applicable, or if both are applicable and produce the same AP value. Algorithm $A$ wins if only it is applicable or if both $A$ and $B$ are applicable and $A$ produces a higher AP value. Algorithm $B$ wins in the symmetric situation.

To address research question RQ1, we execute TARMAQ and each of the 24 adaptive algorithms on the 1100 randomly selected queries from each of the 19 systems. As discussed in Section II each algorithm produces two outcomes: its applicability and, if applicable, an AP value.

Table II shows the resulting data. Note that because TARMAQ can make a recommendation with as little as one relevant transaction, no algorithm is ever applicable when TARMAQ is not. Furthermore, as shown in the table, for 1951 of the 1100 * 19 = 20900 queries, none of the algorithms were able to make a recommendation (i.e., none were applicable).

Looking at the data in the table, the fourth column, $B$ wins, shows that each of the adaptive algorithms has at least some queries for which it outperforms TARMAQ. It might seem unexpected that any of the adaptive techniques would outperform TARMAQ given that TARMAQ has access to ten of thousands of transactions. This occurs when the additional transactions TARMAQ considers muddy the water. For example, given a transaction pairing $a$ and $b$ and the query $a$, the obvious answer is $b$, while given 10000 transactions involving $a$ that don’t all involve $b$ there is less clarity. In summary for RQ1, the adaptive analysis is not completely subsumed by TARMAQ and consequently, viable.

**B. RQ2: Applicability**

RQ2 compares the applicability of the adaptive algorithms to TARMAQ and to each other. Applicability is important to consider because developers will prefer a tool that presents results as often as possible. The range of applicabilities is seen on the $x$-axis of Figure 2. Looking at the relative positions of the points on the $x$-axis, the applicability of the adaptive algorithms covers a wide range. On the right we find the adaptive algorithms that have the same applicability as TARMAQ. Moving to the left, the adaptive techniques have progressively lower applicability. As shown in Table III, applying Tukey’s HSD finds that there is a statistically significant difference in applicability amongst those algorithms that do not match TARMAQ’s applicability except for the pairwise overlap of the four with the lowest applicability (those sharing a common letter are not statistically different).

In summary, compared to each other, as more transactions are required (as the value of $n$ increases) there is a notable drop-off in applicability. This drop-off is considerably sharper when the transactions selected are required to be adjacent. When compared to TARMAQ, 40% of the adaptive algorithms match TARMAQ applicability while the remainder grow progressively less applicable if $n$ increases.

**C. RQ3: Accuracy**

RQ3 compares the MAP values of the adaptive algorithms to TARMAQ and to each other. Attaining a high MAP is key for a recommendation system: the higher an algorithm ranks relevant files and functions, the greater its utility to an engineer. The comparison uses the average precision data obtained from running the 24 adaptive algorithms and TARMAQ on the 20900 queries described in Section VI-B.

We consider two views of the data: overall MAP, which is a harsh measure because it assigns an AP of zero when an algorithm is not applicable, and MAP when applicable, which is a soft measure because it ignores queries to which an algorithm is not applicable.

Table IV presents the results of Tukey’s HSD for the overall MAP values. The values for the adaptive algorithms progressively decrease primarily due to their decreasing applicability. While it is not our expectation that the adaptive algorithms would surpass TARMAQ given the greatly reduced portion of the history they make use of, it is interesting how close some of them come. This suggests, for example, the potential of hybrid techniques. One advantage such hybrids bring is a speed advantage, especially when building efficient on-the-fly recommenders. Evidence supporting the value of such hybrids is seen in the timing comparison shown in Figure 3.

The $y$ axis of Figure 2 shows the range of values for MAP when applicable for the adaptive algorithms and TARMAQ. It is interesting to note that statistically TARMAQ is not superior to any of the adaptive algorithms. In fact numerically, there are four algorithms (dynamic$_{1,n}$adjacent, for $n = 2, 3, 4, 5$) with MAP when applicable values greater than TARMAQ. This means that hybrids such as “apply dynamic$_{1,2}$adjacent if applicable otherwise apply TARMAQ” would match TARMAQ’s applicability while exceeding its MAP value. Overall the “B wins” column of Table II shows how often an adaptive algorithm performs better at the individual query level. While in production we lack an oracle to predict which algorithm to use for a particular query, given the data from the experiments, it is possible to produce the MAP value that a perfect prediction would yield. This value, 0.3798, is a significant improvement over the values produced by any of the individual algorithms.

Thus in summary for RQ3, as expected, none of the adaptive algorithms are a clear replacement for TARMAQ. However, given that they use dramatically less of the history, hybrid approaches deserve consideration. Furthermore, these results also suggest that software is somehow fundamentally different from other domains to which association rule mining has been applied and thus the study of software-specific association rule mining algorithms would be of interest.

**D. RQ4: Adjacency’s Impact**

RQ4 investigates the impact on both applicability and accuracy of requiring transactions to be adjacent. Adjacency of relevant transactions suggests a developer is working on a single issue and thus working in a highly-related portion of the code. Consequently, the information gleaned from adjacent
transactions should be beneficial in identifying key relations between software artifacts.

The investigation makes use of the applicability and average precision data obtained by running 18 adaptive algorithms using the queries described in Section VI-B. We analyze algorithms from the families dynamic, adjacent and dynamic, adjacent in comparison with dynamic. Recall the difference between the two families of adjacent algorithms: dynamic, adjacent uses n adjacent relevant transactions starting from the first relevant transaction, while dynamic, adjacent uses the most recent n adjacent relevant transactions in the history, regardless of where they start. The expectation is that dynamic, adjacent will identify older transactions in exchange for greater applicability.

All three families show a clear power-law reduction in applicability as n increases (the Residual Standard Error values of 0.0717, 0.0008, and 0.0837 for dynamic, adjacent, respectively all indicate good matches to a power-law curve). The applicability of dynamic, adjacent has the least drastic drop-off. For comparison consider an applicability cutoff of one percent. Dynamic maintains viability until approximately n = 2500, while dynamic, adjacent until n = 9, and dynamic, adjacent until only n = 5.

In summary, the three families show a wide range of applicabilities, but a similar pattern, for increasing values of n. The two families that require adjacency exhibit a much more rapid falloff in applicability as n increases. This indicates that large runs of relevant transactions are rare in the change histories. Because of the limited applicability, we focus the accuracy investigation on values of n ranging from 1 to 5.

Figure 4 shows the impact on accuracy caused by requiring transactions to be adjacent. In the figure, a positive difference occurs when adjacency leads to a higher MAP value while a negative difference occurs when adjacency leads to a lower MAP value. For example, the first bar shows that dynamic, adjacent outperforms dynamic, adjacent (by about eight percentage points). Note that for n = 1, dynamic, adjacent, dynamic, adjacent, and dynamic, adjacent are all the same. Thus the two zero-differences at the center of the graph.

Considering first dynamic, adjacent, adjacency yields a clear positive impact on accuracy. This positive impact fluctuates as the value of n increases where there is some indication that by n = 5 the accuracy gap is shrinking. One possible explanation for this pattern is that the inclusion of the first relevant transaction "pins" the age of the first transaction considered. Because this is the same for dynamic, adjacent and dynamic, adjacent, the latter will tend to make use of more recent transactions. The data hints that at some point (e.g., n ≥ 5), the volume of data available tends to counteract the advantage that adjacency brings.

Statistically, there is an interaction that makes it inappropriate to apply Tukey’s HSD test to all the data. Instead we compare dynamic, adjacent and dynamic, adjacent “head to head” for each value of n other than 1 using the Wilcoxon Sum test, which is equivalent to the Mann-Whitney test. The respective p-values are < 0.0001, 0.00052, 0.02433, 0.47. In other words the difference is statistically significant for n = 2, 3, 4, although the difference at 4 is not strong. It is worth noting that for n = 5 the difference is visually smaller and there are only 94 values for dynamic, adjacent, which is less than half of one percent of the 20,900 data points. Both of these factors limit the statistical test’s ability to establish the significance of any difference. Thus, for dynamic, adjacent, we conclude that applying adjacency yields better MAP values at the expense
of applicability, and consequently, $\text{dynamic}_{1,n,\text{adjacent}}$ should be favored over $\text{dynamic}_{1,n}$ when it is applicable.

Finally, we compare the accuracy of $\text{dynamic}_{1,n}$ and $\text{dynamic}_{n,\text{adjacent}}$ where the latter uses the most recent sequence of $n$ adjacent transactions. Because it is not “pinned” to the first relevant transaction, these transactions may be older. In exchange, these algorithms have higher applicability than their $\text{dynamic}_{1,n,\text{adjacent}}$ counterparts. As shown in Figure 4, in this case adjacency has a negative impact on accuracy. From $n = 1$ to $n = 3$ the cost of adjacency grows while for $n > 3$ the negative effect of requiring adjacency diminishes. There are likely two effects here: first, as $n$ increases, $\text{dynamic}_{n,\text{adjacent}}$ searches further back in the history to find $n$ adjacent transactions. Greater age tends to have a negative impact on change impact analysis quality. However the decreasing negative difference in MAP value from $n = 3$ to $n = 5$ indicates that a larger number of adjacent transactions can counter the negative age effect.

Head-to-head statistically comparisons in this case are much stronger as each $p$-value is $< 0.0001$. In other words, the negative impact of requiring adjacency is statistically significant for each value of $n$ (except $n = 1$ where the two algorithms are identical).

In summary for RQ4, adjacency has a significant cost in terms of applicability. This is not unexpected. In exchange, it brings a notable improvement in accuracy, which is mitigated by age.

E. RQ5: Practical Implications

By design, the adaptive algorithms make use of significantly less of the transaction history. This translates into an obvious performance advantage as illustrated in Figure 3. We learned from Figure 2 that some adaptive algorithms, such as $\text{dynamic}_{1,2}$ and $\text{dynamic-P}_{20}$, have equal or only slightly lower applicability than TARMAQ while maintaining
competitive MAP values. Figure 3 illustrates that in exchange there is a dramatic reduction in time it takes to make a recommendation. This is caused by the adaptive techniques considering dramatically fewer transactions (e.g., dynamic$_{1,2}$ and dynamic-P$_{20}$ use only 1.4% and 1.8% of the transactions used by TARMAQ, respectively).

These numbers help us reason about the practical implications of using adaptive algorithms instead of TARMAQ for change impact mining. As alluded to in the Introduction, it generally takes little time to make a recommendation with TARMAQ, provided that the change history is readily available. However, extracting such a change history takes considerable time. For the systems considered in this study, a modestly sized change history of 50,000 transactions takes on average 203 minutes to extract and uses roughly 11MB of disk space per system (after compression). These numbers are no impediment to a normal user, who would only interact with a limited number of active repositories for which the histories would need to be regularly updated (e.g., every night).

However, in the context of providing change impact mining for large numbers of projects, such as for online services like GitHub, the time and space needed to maintain up-to-date histories for all projects becomes unwieldy. The latest report from GitHub (Sept. 2017) claims a total of 67 million repositories, of which 25.3 million are considered active (have seen activity in the preceding year). Keeping a modest (50k) change history for just the active repositories would require 9765 years of extraction time and result in 272 terabytes of data. While incremental creation of the rules would help address the timing challenge, it does not reduce the space needs. Furthermore, this data would need to be frequently updated to keep it timely. Our dynamic approach addresses both aspects. Since adaptive techniques can reduce the time and space required by over 98%, they enable on-the-fly change impact mining of a single project of interest, in contrast to pre-extracting change histories for all projects. In this case, for a single system, impact analysis including the required extraction would take on average only 2:50 minutes and require approximately 150kB. This greatly improves the feasibility of providing change impact mining at this scale, especially for less interactive tasks such as assessing the impact of a pull request in projects that use modern code review.

F. Discussion

Some clear patterns emerge in the data. For example, age seems detrimental to accuracy. In addition, adjacency, while lowering applicability, brings value to the recommendation. These general trends suggest the need to conduct more focused studies considering each of these effects and their interplay.

Furthermore, this initial analysis of adaptive algorithms hints at the complexity of the information in the change history of a software system. For example, when compared to the typical applications of association rule mining, such as analyzing shopping-cart data, the analysis of historical co-change data in a software context looks different. The traditional application of association rule mining aims to leverage “big data.” In contrast, the success of algorithms such as dynamic$_{1,2}$ and dynamic$_{1,5}$ show that when applied to software, using only a few transactions is, at least at times, preferable. While in its preliminary stages, this work pokes at the question “how few transactions are necessary to make good recommendations?”

One final size related observation is hinted at by the top two entries of Table IV. The top performing algorithm is TARMAQ, while second place goes to dynamic-all. What is interesting is that TARMAQ starts with the same relevant transactions as dynamic-all, and then filters them based on overlap with the query. Only transactions with maximal overlap are retained. As shown in the table, this filtering yields a statistically significant improvement in the MAP value. Thus, it would be interesting to investigate, as part of the future work, what happens if TARMAQ’s largest overlap filter step is combined with other adaptive algorithms.

G. Threats to validity

Commits as a basis for evolutionary coupling: The evaluation in this paper is grounded in frequent patterns found in the transactions of change histories. However, these transactions are not in any way guaranteed to be “correct” or “complete” with respect to representing a coherent unit of work [26, 27]. Non-related artifacts may be present, and related artifacts may be missing from a transaction. However, we believe this threat is mitigated in the context of our study, as all but one of the systems (KM) use Git for version control, which promotes coherent transactions with tools for amending commits and rewriting history. For KM, we base transactions on their issue tracking system, which groups relevant commits.

Realism of Scenarios used in Evaluation: Our evaluation establishes a ground truth from historical transactions, randomly splitting them into a query and an expected outcome of a certain size. However, this approach does not account for the actual order in which changes were made before they were committed together to the versioning system. As a result, it is possible that our queries contain elements that were actually changed later in time than elements of the expected outcome. This cannot be avoided when mining co-change data from a versioning system, because the timing of individual changes is lost. It can be addressed by using another source of co-change data, such as a developer’s interactions with an IDE, but the invasiveness of such data collection prevents a study as comprehensive as the one presented here. Moreover, since the evolutionary couplings at the basis of our analysis forms a bi-directional relation, the actual order in which changes were made before they were committed has no impact on the result. Our goal is not to re-enact the actual timeline of changes, but rather to establish a ground truth with respect to related artifacts.

Equal weight for all commits: In our experiments, all transactions from the change history are given equal weight while mining change impact. One could argue that, because of their knowledge about the system, transactions committed by core developers should be given higher weight than transactions committed by occasional contributors. We do not include such weighing scenarios in our study because of their interaction
with several of our research questions. Moreover, most of the systems considered in this study use a modern code review process based on pull-requests to include changes from occasional contributors. We believe this reviewing process largely removes any differences between transactions by core developers and transactions by occasional contributors.

Variation in software systems: We conducted our experiments on two industrial systems and 17 large open source systems that were carefully selected to vary considerably in both system- and change-history characteristics (see Table I). Although this should provide an accurate picture of the adaptive techniques’ performance in various settings, we are likely not to have captured all possible variations.

Implementation: Finally, our prototype ATARI is implemented in Ruby and we conducted the statistical analysis in R. Although we have thoroughly tested our implementations, we can not guarantee the absence of errors that may affect our results.

VIII. RELATED WORK

Silva and Antunes present an in-depth survey of constrained pattern mining [28] in which they describe a range of constraints and properties. Constraint categories include content constraints such as item constraints, value constraints, and aggregate constraints, as well as structural constraints such as length constraints, sequence constraints, and temporal constraints. The adaptive algorithms make use of several of these. For example, adjacency is a sequencing constraint while relevant transactions is a value constraint. An example property is anti-monotonicity: satisfying this property means that whenever an item-set \( X \) violates the constraint, then so does any superset of \( X \). Establishing this property obviates the need to consider supersets of an uninteresting set. The rest of this section traces work on the particular form of constrained pattern mining, association rule mining. Introduced in 1993, the first algorithm for mining association rules was \( \text{AIS} \) (Agrawal-Imielinski-Swami) [20]. Since then, many improvements have been proposed, generally aimed at improving execution time or memory efficiency. These improvements can be classified in four major categories:

1. \textit{Apriori} [29], which uses an efficient pre-computation of rule generation candidates,
2. \textit{Eclat} [30], which partitions the search space into smaller independent subspaces that can be analyzed efficiently,
3. \textit{FPGrowth} [31], which encodes the dataset in a compact tree structure called a frequent patterns tree (FPTree), in order to enable rule mining without having to generate candidate rules, and
4. \textit{ARM} (Rapid Association Rule Mining) [32], which encodes the data set in a prefix-tree ordered by the support of items (SOTrEE).

As an alternative to techniques designed to mine patterns from the complete dataset, targeted association rule mining has been proposed as a refinement which focuses on the generation of rules relevant to a query \([14, 33, 34]\). Targeted association rule mining approaches ignore transactions unrelated to the query, which allows a significant reduction in execution time. One goal of the adaptive algorithms studied in this paper is to yield yet another significant reduction in the number of transactions that need to be considered and consequently a reduction in execution time.

In the specific context of change impact analysis, potentially relevant items are suggested based on \textit{evolutionary (or logical) coupling}. In general, approaches aimed at identifying evolutionary couplings are based upon co-change information, such as those that include coarse- and fine-grained co-changes \([13, 21, 35]\), code-churn \([36]\), and interactions with IDEs \([9]\).

Several projects have considered aspects of the mining problem that, to varying degrees, compliment the investigation of adaptive algorithms. For example, recent research highlighted that the configuration parameters of data mining algorithms have a significant impact on the quality of their results [37]. In the context of association rule mining, several authors have highlighted the need for thoughtfully studying how parameter settings affect the quality of generated rules [38–40]. For example, Moonen et al. recently investigated how the quality of software change recommendation varied depending on association rule mining parameters such as transaction filtering threshold, history length, and history age \([19, 41]\). The interesting question relative to our work concerns the value these ideas bring to adaptive analysis, which often makes a recommendation based on far fewer transactions.

Finally, the software repository mining literature \([15, 19, 42, 43]\) frequently alludes to the notion that learning from a too short, or an overly long history harms the outcome, either because not enough knowledge can be uncovered, or because outdated information introduces noise. Moonen et al. \([19]\) investigated the impact of history size on TARMAQ’s performance. Their discovery that very small histories can yield high quality recommendations was the impetus for our research. Adaptive algorithms bring an intriguing new viewpoint to this discussion.

IX. CONCLUDING REMARKS

Conclusions: When applied to \textit{source-code change impact analysis}, association rules capture implicit knowledge that an engineer has about connections between the artifacts of a system. This paper explores seven families of \textit{adaptive algorithms}. Many of which use dramatically less of the history than existing techniques. Doing so enables us to take a finer-grained look at understanding the value selected transactions bring to the recommendation process. The empirical investigation demonstrated that adaptive algorithms are viable and furthermore that their accuracy can rival that of state-of-the-art \textit{complete-history} techniques, such as TARMAQ.

Contributions: This paper makes the following four contributions:

1. Introduces the notion of \textit{adaptive targeted association rule mining}.
2. Proposes several variants of adaptive algorithms for change impact mining.
3. Implements these algorithms in a prototype tool ATARI.
4. Compares the new algorithms to each other and to the state-of-the-art tool, TARMAQ [10].

Future work: Looking forward, we see several interesting directions for future work. First, there is the combination of an adaptive selection of transactions from the change history with the type of the query-driven selection that is performed...
in TARMQA, where only the transaction(s) covering the largest possible subset of the query are considered for rule generation.

Second, the existence of low applicability and high MAP when applicable algorithms suggests the potential for using machine learning to create an ensemble of algorithms. As a preliminary experiment, a hand-crafted ensemble algorithm was studied, which applies dynamic1Adjacent if it is applicable, and uses TARMQA otherwise. This ensemble takes advantage of dynamic1Adjacent’s high MAP while maintaining TARMQA's high applicability. The ensemble increased the MAP value while maintaining the same high applicability.

ACKNOWLEDGEMENTS

We would like to thank Thomas Rolfsnes for his guidance in working with the TARMQA implementation, and thank Cisco Norway and Kongsberg Maritime for sharing their data.

REFERENCES


[31] J. Han, J. Pei, Y. Yin, and R. Mao, “Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach,” Data Mining and Knowledge Discovery, vol. 8, no. 1, pp. 53–87, 2004.


