2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation

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 a software system grows, it becomes increasingly difficult for developers to be aware of all the dependencies that exist between artifacts (e.g., files or methods) of the system. Change impact analysis helps to overcome this problem, as it recommends to a developer relevant source-code artifacts related to her current 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 focuses on only those transactions potentially *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 association rule mining that considers a *dynamic selection* of the relevant transactions. It can be viewed as a 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 necessary groundwork for the future study of adaptive techniques and their application to challenges such as the *on-the-fly* style of impact analysis that is needed 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 manifests itself through, e.g., commits and their comments [11], bug reports and their 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 variations of *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 TARMAQ 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.

This paper proposes and explores a new approach, ATARI *(Adaptive Targeted Association Rule mIning)*, and empirically investigates 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 has the potential to 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., 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, our experiments suggest that software is somehow fundamentally different and thus warrants future work on software-specific mining variants. To be clear, our goal is more subtle than a straight-forward attempt to "provide a better recommendation." While better clearly brings value, a new approach that produces 80% of the answer using only 20% of the resources is also of great interest because of its potential to lead to even better algorithms down the road.

Second, adaptive techniques open up the possibility for integration of (within-project) change impact mining with online services such as GitHub. In general, making a recommendation is fast, even when using the entire relevant history. However, *extracting* the required history takes considerably longer. For a service like GitHub, the space and time required to keep up-to-date extracted histories for all active projects is prohibitive, making the alternative of on-the-fly adaptive analysis preferable.

This paper makes the following contributions:

- it introduces the concept of *adaptive targeted association rule mining*,

2470-6892/18/$31.00 ©2018 IEEE
DOI 10.1109/SCAM.2018.00022

- it proposes several adaptive algorithms for change impact mining,
- it studies the implementation of these algorithms in the prototype tool, ATARI, and finally
- 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 to change impact mining in Sections 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 *bread* $\rightarrow$ *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 $\mathcal{T}$, where each transaction originates with a commit from a versioning system. More specifically, a transaction $T \in \mathcal{T}$ 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] restricts the generation of rules using a constraint, which 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 that share one of these artifacts would need to be considered.

Definition 1 (Relevant Transaction): Given a query q, a transaction $t \in \mathcal{T}$ 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 the 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, then 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 a ranked impact list 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. AP is computed as the average of Precision@k where k is the rank of each artifact in I_q that is found in E_q.

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

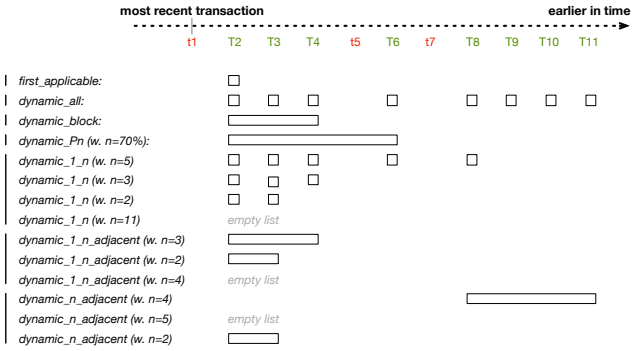

Fig. 1. A visualization of the various adaptive algorithms

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-changed with the largest possible subset of the query. This constraint balances the precision of a complete match with the applicability that comes from exploiting partial matches.

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 thus 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: $\mathcal{T} = t_1, T_2, T_3, T_4, t_5, T_6, t_7, T_8, T_9, T_{10}, T_{11}$, 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 TARMAQ algorithm [10].

The simplest adaptive algorithm, *first-applicable*, selects the first relevant transaction. In the example this is transaction T_2. Like all the adaptive algorithms, first-applicable begins its search from the most recent transaction. 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_6, 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 conscious 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_6, T_8. 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*$_{1,n}$*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*$_{1,2}$*adjacent* yields T_2, T_3, while *dynamic*$_{1,5}$*adjacent* yields the empty list. We consider the five *dynamic*$_{1,n}$*adjacent* family members for $n \in \{1, 2, 3, 4, 5\}$.

The last family, *dynamic*$_n$*adjacent*, aims to increase *dynamic*$_{1,n}$*adjacent*'s low applicability while preserving its high average precision. *Dynamic*$_n$*adjacent* uses the most recent n adjacent relevant transactions. It differs from *dynamic*$_{1,n}$*adjacent* in that it is not anchored at the first relevant transaction. For example, *dynamic*$_4$*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*$_n$*adjacent* family members for $n \in \{1, 2, 3, 4, 5\}$.

V. RESEARCH QUESTIONS

To better understand the pros and cons of using adaptive algorithms for mining association rules, we consider the following research questions:

RQ1 **Viability:** Are there queries for which the adaptive algorithms' performance is comparable to TARMAQ?

RQ2 **Applicability:** How does the applicability of the adaptive algorithms compare to each other and to that of TARMAQ? (This includes inter-family comparisons 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 TARMAQ? *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 GitHub-scale recommendations?

As mentioned earlier, our goal is not to simply produce a better change impact mining algorithm, but to better understand the interplay between the number of transactions considered and the quality of the recommendation possible. For example, the first research question would be negatively answered if the adaptive algorithms *never* performed comparable to TARMAQ. On the other hand, the more often they do so, the greater the possibility of exploiting them in a hybrid algorithm.

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 the changes to code that is not part of a function (e.g., global variable declarations).

TABLE I
CHARACTERISTICS OF THE SOFTWARE SYSTEMS STUDIED (BASED ON OUR EXTRACTION OF THE MOST RECENT 50 000 TRANSACTIONS FOR EACH).

System	History (years)	Unique files	Unique artifacts	Languages used*
CPython	12.05	7725	30090	Python, C, 16 others
Gecko	1.08	86650	231850	C++, C, JavaScript, 34 others
Git	11.02	3753	17716	C, shell script, Perl, 14 others
Hadoop	6.91	24607	272902	Java, XML, 10 others
HTTPD	19.78	10019	29216	XML, C, Forth, 19 others
IntelliJ IDEA	2.61	62692	343613	Java, Python, XML, 26 others
Liferay Portal	0.87	144792	767955	Java, XML, 12 others
Linux Kernel	0.77	26412	161022	C, 16 others
LLVM	4.54	25600	66604	C++, Assembly, C, 16 others
MediaWiki	11.69	12252	12267	PHP, JS, 11 others
MySQL	10.68	42589	136925	C++, C, JS, 24 others
PHP	10.82	21295	53510	C, PHP, XML, 24 others
Ruby on Rails	11.42	10631	10631	Ruby, 6 others
RavenDB	8.59	29245	139415	C#, JS, XML, 12 others
Subversion	14.03	6559	46136	C, Python, C++, 15 others
WebKit	3.33	281898	397850	HTML, JS, C++, 24 others
Wine	6.6	8234	126177	C, 16 others
Cisco	2.43	64974	251321	C++, C, C#, Python, others
KM	15.97	35111	35111	C++, C, XML, others

* languages used by open source systems from http://www.openhub.net.

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.[1] 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 the prototype tool ATARI (Adaptive Targeted

[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.

Association Rule mIning). ATARI was developed in RUBY, and built as a fork of the TARMAQ 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 considers the viability of the adaptive approach. Given that TARMAQ 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, 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 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. For 1951 of the $1100 * 19 = 20\,900$ queries, none of the algorithms were able to make a recommendation (i.e., none were applicable).

The fourth column, "B wins", in Table II 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 out-perform TARMAQ given that TARMAQ has access to tens 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 when given 10 000 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's applicability while the remainder grow progressively less applicable if n increases.

TABLE II
VIABILITY DATA FOR THE 24 ADAPTIVE ALGORITHMS

Algorithm B	TARMAQ wins	AP tie	B wins	B not applicable
first-applicable	7551	8125	3273	0
dynamic-all	3393	11781	3775	0
dynamic-block	7398	8132	3419	0
dynamic-P_{20}	7199	8124	3626	0
dynamic-P_{50}	7321	8114	3514	0
dynamic-P_{80}	7404	8108	3437	0
dynamic$_{1,1}$	7551	8125	3273	0
dynamic$_{1,2}$	6315	6515	4143	1976
dynamic$_{1,3}$	5567	5648	4460	3274
dynamic$_{1,4}$	5089	5142	4505	4213
dynamic$_{1,5}$	4770	4774	4472	4933
dynamic$_{1,10}$	3803	3653	4158	7335
dynamic$_{1,100}$	1460	1092	1771	14626
dynamic$_{1,1000}$	158	95	112	18584
dynamic$_{1,1}$*adjacent*	7551	8125	3273	0
dynamic$_{1,2}$*adjacent*	645	738	493	17073
dynamic$_{1,3}$*adjacent*	146	195	171	18437
dynamic$_{1,4}$*adjacent*	57	68	75	18749
dynamic$_{1,5}$*adjacent*	28	33	33	18855
dynamic$_1$*-adjacent*	7551	8125	3273	0
dynamic$_2$*-adjacent*	5850	3453	1898	7748
dynamic$_3$*-adjacent*	3371	1757	908	12913
dynamic$_4$*-adjacent*	1629	893	471	15956
dynamic$_5$*-adjacent*	864	522	245	17318

TABLE III
TUKEY'S HSD FOR APPLICABILITY

Algorithm	Appl.	Group
dynamic$_{1,1}$	0.9066	a
dynamic$_{1,1}$*adjacent*	0.9066	a
dynamic$_1$*-adjacent*	0.9066	a
dynamic-all	0.9066	a
dynamic-block	0.9066	a
dynamic-P_{20}	0.9066	a
dynamic-P_{50}	0.9066	a
dynamic-P_{80}	0.9066	a
first-applicable	0.9066	a
TARMAQ	0.9066	a
dynamic$_{1,2}$	0.8121	b
dynamic$_{1,3}$	0.7500	c
dynamic$_{1,4}$	0.7050	d
dynamic$_{1,5}$	0.6706	e
dynamic$_{1,10}$	0.5556	f
dynamic$_2$*-adjacent*	0.5359	g
dynamic$_3$*-adjacent*	0.2888	h
dynamic$_{1,100}$	0.2068	i
dynamic$_4$*-adjacent*	0.1432	j
dynamic$_{1,2}$*adjacent*	0.0897	k
dynamic$_5$*-adjacent*	0.0780	l
dynamic$_{1,3}$*adjacent*	0.0244	m
dynamic$_{1,1000}$	0.0174	mn
dynamic$_{1,4}$*adjacent*	0.0095	no
dynamic$_{1,5}$*adjacent*	0.0044	o

TABLE IV
TUKEY'S HSD FOR OVERALL MAP

Algorithm	MAP	Group
TARMAQ	0.2487	a
dynamic-all	0.2362	b
dynamic-P_{20}	0.2260	c
dynamic-P_{50}	0.2246	c
dynamic-block	0.2233	cd
dynamic-P_{80}	0.2233	cd
dynamic$_{1,1}$	0.2209	cd
dynamic$_{1,1}$*adjacent*	0.2209	cd
dynamic$_1$*-adjacent*	0.2209	cd
first-applicable	0.2209	cd
dynamic$_{1,2}$	0.2139	d
dynamic$_{1,3}$	0.2002	e
dynamic$_{1,4}$	0.1875	f
dynamic$_{1,5}$	0.1767	g
dynamic$_{1,10}$	0.1414	h
dynamic$_2$*-adjacent*	0.1091	i
dynamic$_3$*-adjacent*	0.0527	j
dynamic$_{1,100}$	0.0444	j
dynamic$_4$*-adjacent*	0.0281	k
dynamic$_{1,2}$*adjacent*	0.0276	k
dynamic$_5$*-adjacent*	0.0166	l
dynamic$_{1,3}$*adjacent*	0.0074	lm
dynamic$_{1,1000}$	0.0040	m
dynamic$_{1,4}$*adjacent*	0.0029	m
dynamic$_{1,5}$*adjacent*	0.0012	m

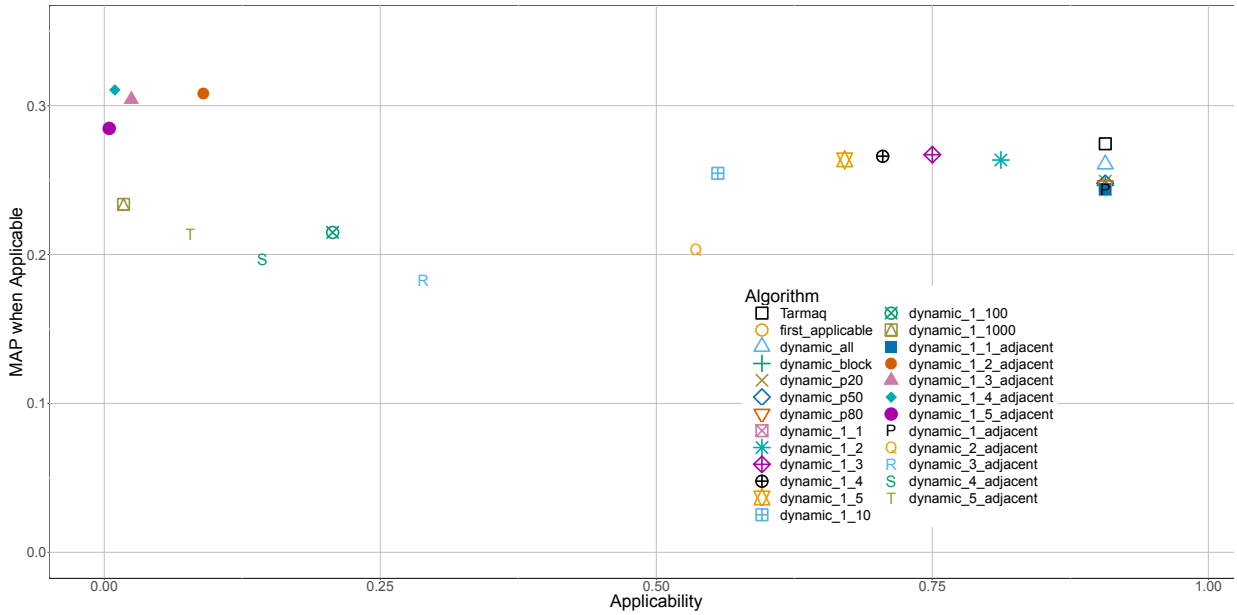

Fig. 2. A visualization of MAP and applicability for the adaptive algorithms and TARMAQ. The overlap makes discerning the individual algorithms difficult. Instead the figure is aimed at illustrating the general trend related to MAP values and applicability.

C. RQ3: Accuracy

RQ3 compares the MAP values of the adaptive algorithms to TARMAQ and to each other. Attaining a high MAP value 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 20 900 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 *MAP when applicable* values 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's. 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 compute 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

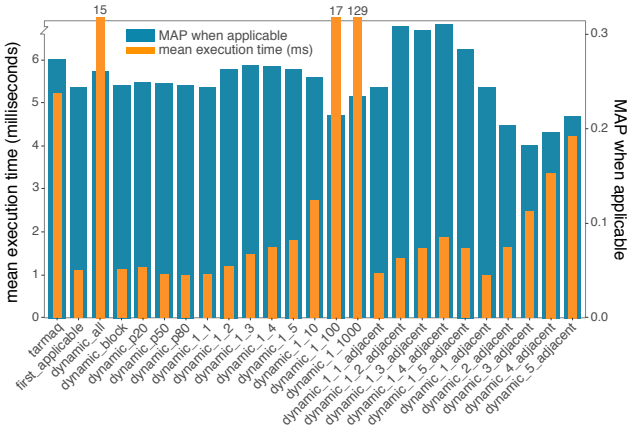

Fig. 3. Mean execution time of TARMAQ and the adaptive algorithms, together with the MAP value attained by each. Three outliers were truncated to keep a legible scale, their values are shown at the top of the plot.

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. Thus, the information gleaned from adjacent transactions should identify 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*$_{1,n}$*adjacent* and *dynamic*$_n$*adjacent* in comparison with *dynamic*$_{1,n}$. Recall the difference between the two families of adjacent algorithms: *dynamic*$_{1,n}$*adjacent* uses n adjacent relevant transactions *starting from the first relevant transaction*, while *dynamic*$_n$*adjacent* uses the most recent n adjacent relevant transactions in the history, regardless of where they start. The expectation is that *dynamic*$_n$*adjacent* will identify older transactions in exchange for greater applicability.

To begin with, all three families show a clear power-law reduction in applicability as n increases (with Residual Standard Error values of 0.0717 for *dynamic*$_{1,n}$, 0.0008 for *dynamic*$_{1,n}$*adjacent*, and 0.0837 for *dynamic*$_n$*adjacent*). The applicability of *dynamic*$_{1,n}$ has the least drastic drop-off. For comparison, consider an applicability cut-off of one percent. *Dynamic*$_{1,n}$ is viable until approximately $n = 2500$, while *dynamic*$_n$*adjacent* until $n = 9$, and *dynamic*$_{1,n}$*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*$_3$ outperforms *dynamic*$_3$*adjacent* (by about eight percentage points). Note that for $n = 1$, *dynamic*$_n$, *dynamic*$_n$*adjacent*, *dynamic*$_{1,n}$, and *dynamic*$_{1,n}$*adjacent* are all the same. Thus the two zero-differences at the center of the graph.

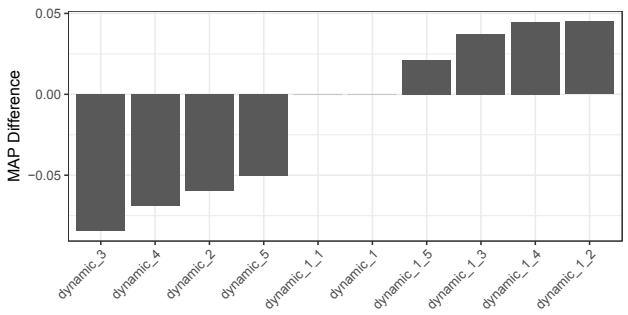

Fig. 4. Difference in MAP-when-Applicable When Adjacency is Required

Considering first *dynamic*$_{1,n}$, adjacency yields a clear positive impact on accuracy. The impact fluctuates as the value of n increases, with an indication that by $n = 5$ the accuracy gap is shrinking. One possible explanation is that inclusion of the first relevant transaction "pins" the age of the first transaction considered. Because this is the same for *dynamic*$_{1,n}$ and *dynamic*$_{1,n}$*adjacent*, the latter will use more recent transactions. At some point (e.g., $n \geq 5$), the volume of data counteracts 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*$_{1,n}$*adjacent* and *dynamic*$_{1,n}$ "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 significance 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*$_{1,5}$*adjacent*, which is less than half of one percent of the 20 900 data points considered. Both of these factors limit the statistical test's ability to establish the significance of any difference. Thus, for *dynamic*$_{1,n}$, we conclude that applying adjacency yields better MAP values at the expense of applicability, and consequently, *dynamic*$_{1,n}$*adjacent* should be favored over *dynamic*$_{1,n}$ when it is applicable.

Finally, we compare the accuracy of *dynamic*$_{1,n}$ and *dynamic*$_n$*adjacent* where the latter uses the most recent sequence of n adjacent transactions. Because it is not "pinned" to the first relevant transaction, the transactions involved are likely older. In exchange, these algorithms have higher applicability than their *dynamic*$_{1,n}$*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, *dynamic*$_n$*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 statistical comparisons in this case are much stronger as each p-value is < 0.0001. 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 notably improves accuracy, which is dampened by age.

E. RQ5: Practical Implications

By design, the adaptive algorithms make use of significantly less of the transaction history. In the extreme (i.e., a task that represents the worst case for TARMAQ), this difference translates into an obvious performance advantage as illustrated

in Figure 3. We learned from Figure 2 that some adaptive algorithms, such as *dynamic*$_{1,2}$ and *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 the 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 of CPU time to extract and uses roughly 11MB of disk space per system after compression. These numbers are no impediment to a normal user who only interacts with a limited number of active repositories, making periodic (e.g., nightly) updates of the history viable.

However, in the context of providing (within-project) 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 CPU time for initial extraction and result in 272 terabytes of compressed data. While incremental extraction of new transactions will help with keeping the histories up-to-date, it will not solve the initial extraction effort, nor will it address space requirements. The extraction process can of course be parallelized, but it takes a large number of cores to turn 9765 years of CPU-time into a feasible time-span (e.g., it would take 507.780 cores to reduce extraction time to a week, without accounting for increased communication overhead and bandwidth challenges).

Our adaptive 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 the 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 what happens when 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

The first algorithm for mining association rules was introduced in 1993, *AIS* (Agrawal-Imielinksi-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) *Apriori* [28], which uses an efficient pre-computation of rule generation candidates, (2) *Eclat* [29], which partitions the search space into smaller independent subspaces that can be analyzed efficiently, (3) *FPGrowth* [30], 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) RARM (Rapid Association Rule Mining) [31], which encodes the data set in a prefix-tree ordered by the support of items (SOTrieIT). In addition, evolution of the dataset from which association rules are mined (e.g., the addition of new transactions) has motivated *incremental association rule mining* [32], which aims to *update* the earlier mined rules based on the changes to the dataset.

As a refinement to techniques that mine *all* patterns in a dataset, *targeted* association rule mining constrains rule generation (i.e., pattern mining) to those relevant to a query [14]. Targeted association rule mining ignores transactions unrelated to the query, which significantly reduces execution time. The adaptive algorithms studied in this paper aim at an additional significant reduction in the number of transactions considered, and consequently at a reduction in execution time.

Silva and Antunes present an in-depth survey of constrained pattern mining [33] 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 constraints. For example, adjacency is a sequencing constraint while relevant transactions is a value constraint.

In the specific context of change impact analysis, potentially relevant items are suggested based on *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, 34], code-churn [35], 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 [36]. 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 [37–39]. 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, 40]. 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, 41, 42] 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 *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 *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 *complete-history* techniques such as TARMAQ.

Contributions: This paper makes the following four contributions: (1) Introduces the notion of 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. The first applies TARMAQ as a filter to the transactions selected by an adaptive algorithm. Second, it might be interesting to attempt to provide a natural language explanation supporting each recommendation.

Finally, 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 $dynamic_{1,2}adjacent$ if it is applicable, and uses TARMAQ otherwise. This ensemble takes advantage of $dynamic_{1,2}adjacent$'s high MAP while maintaining TARMAQ'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 TARMAQ implementation, and thank Cisco Norway and Kongsberg Maritime for sharing their data.

REFERENCES

[1] S. Bohner and R. Arnold, *Software Change Impact Analysis*. CA, USA: IEEE, 1996.

[2] J. Law and G. Rothermel, "Whole Program Path-Based Dynamic Impact Analysis," in *Int'l Conf. Softw. Engineering*. IEEE, 2003, pp. 308–318.

[3] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley, "Chianti: a tool for change impact analysis of java programs," in *Conf. Object-oriented Programming, Systems, Languages, and Applications*, 2004, pp. 432–448.

[4] M.-A. Jashki, R. Zafarani, and E. Bagheri, "Towards a more efficient static software change impact analysis method," in *Ws. Program Analysis for Softw. Tools and Engineering (PASTE)*. ACM, 2008, pp. 84–90.

[5] A. R. Yazdanshenas and L. Moonen, "Crossing the boundaries while analyzing heterogeneous component-based software systems," in *Int'l Conf. Softw. Maintenance*. IEEE, 2011, pp. 193–202.

[6] A. Podgurski and L. Clarke, "A formal model of program dependences and its implications for software testing, debugging, and maintenance," *IEEE TSE*, vol. 16, no. 9, pp. 965–979, 1990.

[7] A. E. Hassan and R. Holt, "Predicting change propagation in software systems," in *Int'l Conf. Softw. Maintenance*. IEEE, 2004, pp. 284–293.

[8] G. Canfora and L. Cerulo, "Impact Analysis by Mining Software and Change Request Repositories," in *Int'l Softw. Metrics Symp.* IEEE, 2005, pp. 29–37.

[9] M. B. Zanjani, G. Swartzendruber, and H. Kagdi, "Impact analysis of change requests on source code based on interaction and commit histories," in *Int'l Working Conf. Mining Softw. Repositories*, 2014, pp. 162–171.

[10] T. Rolfsnes, S. Di Alesio, R. Behjati, L. Moonen, and D. W. Binkley, "Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis," in *Int'l Conf. Softw. Analysis, Evolution, and Reengineering*. IEEE, 2016, pp. 201–212.

[11] S. Eick, T. L. Graves, A. Karr, J. Marron, and A. Mockus, "Does code decay? Assessing the evidence from change management data," *IEEE TSE*, vol. 27, no. 1, pp. 1–12, 2001.

[12] M. Gethers, H. Kagdi, B. Dit, and D. Poshyvanyk, "An adaptive approach to impact analysis from change requests to source code," in *Int'l Conf. Automated Softw. Engineering*. IEEE, 2011, pp. 540–543.

[13] R. Robbes, D. Pollet, and M. Lanza, "Logical Coupling Based on Fine-Grained Change Information," in *Working Conf. Reverse Engineering*. IEEE, 2008, pp. 42–46.

[14] R. Srikant, Q. Vu, and R. Agrawal, "Mining Association Rules with Item Constraints," in *Int'l Conf. Knowledge Discovery and Data Mining (KDD)*. AASI, 1997, pp. 67–73.

[15] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *IEEE TSE*, vol. 31, no. 6, pp. 429–445, 2005.

[16] A. T. T. Ying, G. Murphy, R. T. Ng, and M. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE TSE*, vol. 30, no. 9, pp. 574–586, 2004.

[17] H. Kagdi, S. Yusuf, and J. I. Maletic, "Mining sequences of changed-files from version histories," in *Int'l Ws. Mining Softw. Repositories*. ACM, 2006, pp. 47–53.

[18] A. Alali, "An Empirical Characterization of Commits in Software Repositories," Ms.c, Kent State University, 2008.

[19] 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.

[20] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *Int'l Conf. Management of Data*. ACM, 1993, pp. 207–216.

[21] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *Int'l Conf. Softw. Maintenance*. IEEE, 1998, pp. 190–198.

[22] R. Baeza-Yates and B. Ribeiro-Neto, *Modern information retrieval*. ACM, 1999.

[23] T. Rolfsnes, L. Moonen, S. D. Alesio, R. Behjati, and D. Binkley, "Aggregating Association Rules to Improve Change Recommendation," *Empirical Software Engineering (EMSE)*, vol. 23, no. 2, pp. 987–1035, 2018.

[24] M. L. Collard, M. J. Decker, and J. I. Maletic, "srcML: An Infrastructure for the Exploration, Analysis, and Manipulation of Source Code: A Tool Demonstration," in *Int'l Conf. Softw. Maintenance*. IEEE, 2013, pp. 516–519.

[25] H. Kagdi, M. Gethers, and D. Poshyvanyk, "Integrating conceptual and logical couplings for change impact analysis in software," *Empirical Software Engineering*, vol. 18, no. 5, pp. 933–969, 2013.

[26] K. Herzig and A. Zeller, "The impact of tangled code changes," in *Working Conf. Mining Softw. Repositories*. IEEE, 2013, pp. 121–130.

[27] K. Herzig, S. Just, and A. Zeller, "The impact of tangled code changes on defect prediction models," *Empirical Software Engineering*, vol. 21, no. 2, pp. 303–336, 2016.

[28] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," in *Int'l Conf. Very Large Data Bases (VLDB)*, 1994, pp. 487–499.

[29] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 3, pp. 372–390, 2000.

[30] 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.

[31] A. Das, W.-K. Ng, and Y.-K. Woon, "Rapid association rule mining," in *Proceedings of the tenth Int'l Conf. Information and knowledge management - CIKM'01*. New York, New York, USA: ACM Press, 2001, p. 474.

[32] B. Nath, D. K. Bhattacharyya, and A. Ghosh, "Incremental association rule mining: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 3, no. 3, pp. 157–169, 2013.

[33] A. Silva and C. Antunes, "Constrained pattern mining in the new era," *Knowl. Inf. Syst.*, vol. 47, no. 3, Jun. 2016.

[34] D. Beyer and A. Noack, "Clustering Software Artifacts Based on Frequent Common Changes," in *Int'l Ws. Program Comprehension*. IEEE, 2005, pp. 259–268.

[35] H. Gall, M. Jazayeri, and J. Krajewski, "CVS release history data for detecting logical couplings," in *Int'l Ws. Principles of Softw. Evolution (IWPSE)*. IEEE, 2003, pp. 13–23.

[36] O. Maimon and L. Rokach, *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds. Springer, 2010.

[37] Z. Zheng, R. Kohavi, and L. Mason, "Real world performance of association rule algorithms," in *Int'l Conf. Knowledge Discovery and Data Mining (KDD)*. ACM, 2001, pp. 401–406.

[38] W. Lin, S. A. Alvarez, and C. Ruiz, "Efficient Adaptive-Support Association Rule Mining for Recommender Systems," *Data Mining and Knowledge Discovery*, vol. 6, no. 1, pp. 83–105, 2002.

[39] N. Jiang and L. Gruenwald, "Research issues in data stream association rule mining," *ACM SIGMOD Record*, vol. 35, no. 1, pp. 14–19, 2006.

[40] L. Moonen, S. Di Alesio, D. Binkley, and T. Rolfsnes, "Practical guidelines for change recommendation using association rule mining," in *Int'l Conf. Automated Softw. Engineering*. ACM, 2016, pp. 732–743.

[41] T. L. Graves, A. Karr, J. Marron, and H. P. Siy, "Predicting fault incidence using software change history," *IEEE TSE*, vol. 26, no. 7, pp. 653–661, 2000.

[42] A. E. Hassan, "The road ahead for Mining Software Repositories," in *Frontiers of Softw. Maintenance*. IEEE, 2008, pp. 48–57.