The Journal of Systems and Software 164 (2020) 110550

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

On Adaptive Change Recommendation

Leon Moonen[a,*], David Binkley[b,*], Sydney Pugh[b]

[a] Simula Research Laboratory, Oslo, Norway
[b] Loyola University Maryland, 4501 N. Charles St., Baltimore, MD 21210-2699, USA

ARTICLE INFO

Article history:
Received 18 March 2019
Revised 15 January 2020
Accepted 11 February 2020
Available online 12 February 2020

Keywords:
Change impact analysis
Change recommendation
Evolutionary coupling
Association rule mining

ABSTRACT

As the complexity of a software system grows, it becomes harder for developers to be aware of all the dependencies between its artifacts (e.g., files or methods). Change impact analysis helps to overcome this challenge, by recommending relevant source-code artifacts related to a developer's 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 typically uses a change history of tens of thousands of transactions. For efficiency, *targeted* association rule mining constrains the transactions used to those potentially *relevant* to answering a particular *query*. However, it still considers *all* the relevant transactions in the history.

This paper presents ATARI, a new *adaptive* approach that further constrains targeted association rule mining by considering a *dynamic selection* of the relevant transactions. Our investigation of *adaptive change impact mining* empirically studies fourteen 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 this direct comparison, our investigation motivates and lays the groundwork for the future study of adaptive techniques, and their application to challenges such as *on-the-fly* impact analysis at GitHub-scale.

1. 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* (Bohner and Arnold, 1996) has been proposed as a solution to this problem. Traditionally, techniques for change impact analysis have used static or dynamic analysis to identify dependencies, for example, methods calling or called by a changed method (Law and Rothermel, 2003; Ren et al., 2004; Jashki et al., 2008). However, static and dynamic analysis are generally language-specific, making them hard to apply to modern heterogeneous software systems (Yazdanshenas and Moonen, 2011). 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 (Podgurski and Clarke, 1990).

To address these challenges, alternative techniques have been proposed that identify dependencies through *evolutionary coupling* (Hassan and Holt, 2004; Canfora and Cerulo, 2005; Zanjani et al., 2014; Rolfsnes et al., 2016). In essence, these techniques exploit a developer's inherent knowledge of dependencies in the system, which manifests itself through, for example, commits and their comments (Eick et al., 2001), bug reports and their fixes (Gethers et al., 2011), and IDE activity (Robbes et al., 2008). Evolutionary couplings differ from those found through static and dynamic analysis, because they are based on how the software system has evolved over time, rather than how system components are interconnected. The resulting recommendation is a list of relevant source-code artifacts for the engineer to consider.

This paper exploits *historical co-change* between artifacts, that is artifacts that have changed together in history, as the basis for uncovering evolutionary coupling. It does this using variations of *targeted association rule mining* (Srikant et al., 1997). We refer to the process of using mined evolutionary couplings for change impact analysis as *change impact mining*.

Existing algorithms for change impact mining (Zimmermann et al., 2005; Ying et al., 2004; Kagdi et al., 2006; Alali, 2008; Rolfsnes et al., 2016) consider the *complete* set of transactions in the change history. Recent work with one of these algorithms, TARMAQ,

* Corresponding authors.
E-mail addresses: leon.moonen@computer.org (L. Moonen), binkley@cs.loyola.edu (D. Binkley), sfpugh@loyola.edu (S. Pugh).

https://doi.org/10.1016/j.jss.2020.110550

on the impact of history length on analysis quality (Moonen et al., 2018) found 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 consider.

The goal of such algorithms is to aid an engineer working with the code. Thus a good recommendation includes those artifacts that the engineer needs to be aware of. For example, the two most common use cases involve an engineer faced with a maintenance request. In the first, the engineer has planned an update to certain parts of the code and wants to know if there are any missing parts. In the second use case, the engineer has made a fix and the tool can automatically provide suggestions to the engineer regarding potentially missed changes.

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 (Srikant et al., 1997) akin to how targeted association rule mining improved on association rule mining (Agrawal et al., 1993). Thus beyond the immediate performance comparison, we hope that our study of adaptive change impact mining will inspire others to investigate the impact of software-specific adaptive targeted association rule mining techniques.

Our motivation for studying adaptive techniques is two-fold: first, we seek to better understand the interplay between the transaction history and change impact mining. 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, our experiments suggest that software is somehow fundamentally different and thus warrants research on software-specific association rule 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 interest because of its potential to lead to even better, more resource-intensive, 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, however, the work presented here shows that on-the-fly adaptive analysis is a viable alternative!

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 implementations of these algorithms in the prototype tool, ATARI,
- it compares the new algorithms to each other and to the state-of-the-art, TARMAQ (Rolfsnes et al., 2016), and finally
- it considers the impact of applying TARMAQ's rule selection to the adaptively chosen rules.

This article refines and extends our preliminary results (Pugh, et al., 2018) by providing a more in-depth explanation of the adaptive algorithms, extending the original analysis to include twice as many algorithms, and adding the investigation into incorporating TARMAQ's rule selection, which nearly doubles the number of research questions considered.

The remainder of the paper provides background on association rule mining and its application to change impact mining in Sections 2 and 3. The newly devised adaptive techniques are presented in Section 4, followed by their study in Sections 5 through 8. The paper concludes with a discussion of related work in Section 9 and a summary in Section 10.

2. Background on 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 (Agrawal et al., 1993). Readers familiar with these concepts should skim this section to become familiar with the notation used in the remainder of the paper. *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. For version control systems that support side *branches*, we consider only the main branch and thus only take the changes from side branches into account after these are merged back into the main branch. Within the history, a transaction $T \in \mathcal{T}$ is the set of artifacts (e.g., files or functions) that were either changed or added while addressing a given bug fix or feature addition, hence creating a *logical dependence* between the artifacts (Gall et al., 1998). In contrast to applications outside of software engineering, where the history is treated as a set, we treat the history as an age ordered sequence and define two transactions as *adjacent* if they occur one right after the other in the history.

As originally defined (Agrawal et al., 1993), association rule mining makes use of the complete history. *Targeted association rule mining* (Srikant et al., 1997) restricts the transactions used by applying a *constraint*. An example constraint specifies that the artifacts of selected transactions must belong to a particular set of interest. Doing so dramatically reduces the number of rules generated and thus improves rule-generation time (Srikant et al., 1997). The *adaptive* algorithms considered in this paper provide a similar dramatic reduction in the number of transactions considered. Thus this paper lays groundwork parallel to that of the initial paper on targeted association rule mining.

Change impact mining involves four steps: (1) select the transactions from which to build rules (based on certain constraints), (2) generate rules from the selected transactions, (3) rank the rules, and (4) provide a recommendation based on the highest-ranked rules. In the experiments, TARMAQ's default configuration is used, except regarding history filtering, which we describe in Section 6.2. Step 1, transaction selection, 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. We define such transactions as *relevant:*

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.)

If the selection, Step 1, yields an empty list, then no recommendation can be given (e.g., given a query of artifacts not found anywhere in the history, it is impossible to select transactions from which to build rules). Alternatively, when the selection produces a non-empty list, then we define the algorithm as *applicable* to the query. Otherwise, we say that the algorithm is *not applicable* to the query. Other things being equal, an algorithm with higher applicability, one that is applicable to more queries, 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 one or more interestingness measures (Tan et al., 2002; Rolfsnes et al., 2017) such as *support* and *confidence* (Agrawal et al., 1993). 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. Based on prior research, we do not make use of (arbitrary) cutoff values for support and confidence, instead, rules are ranked based on support, breaking ties using confidence (Agrawal et al., 1993).

Finally, transactions do not record the order of the individual changes involved. Hence, in the evaluation, we empirically assess the quality of a change 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 serves as the ground truth. This process mimics a developer in the process of making the change covered by t, but forgetting one or more artifacts (i.e., those of E_q). This splitting approach, which yields a uniform distribution of query and expected-outcome sizes, is standard in the evaluation of change impact mining techniques (Zimmermann et al., 2005; Ying et al., 2004; Kagdi et al., 2006; Alali, 2008; Rolfsnes et al., 2016). Furthermore, we use only those transactions from the history that are older than t, to construct the association rules based on query q. The result is a ranked impact list I_q of potentially impacted artifacts from the history.

3. 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 (Zimmermann et al., 2005) and FP-tree, which employs FP-trees to provide change recommendation (Ying et al., 2004), were independently developed around the same time. Both Rose and FP-tree only uncover artifacts that changed in the history together with *all* elements of the query. This constraint aims to ensure a high-quality recommendation, which, however, limits the applicability of these algorithms. Indeed, empirical evaluation shows that Rose and FP-tree are unable to make a recommendation more often than not (Rolfsnes et al., 2016).

At the other end of the spectrum the Co-Change algorithm (Kagdi et al., 2006) 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 (Rolfsnes et al., 2016) attempts to balance these two. It uncovers the artifacts that have co-changed with the largest possible subset of the query. This transaction selection constraint balances the precision of a complete match with the applicability that comes from exploiting partial matches. In cases where a complete match is possible, Rose, FP-tree, and Tarmaq all yield the same result. However, in cases where only a subset of the query can be matched, Rose and FP-tree fail to provide a recommendation, whereas Tarmaq produces a recommendation based on the largest partial overlap between the query and the transactions of the change history. In comparison with Co-Change, Tarmaq's matching of transactions with the largest partial overlap with the query, reduces noise, provided this overlap is greater than size one. If the overlap is only one element, Tarmaq effectively behaves the same as Co-Change.

4. Adaptive techniques

This section introduces the eight families of adaptive techniques considered in our study. We refer to them as *families* because the latter five are parameterized and thus give rise to multiple algorithms (the first three are unparameterized, i.e., families of one). To illustrate each family, Fig. 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.

As mentioned in Section 2, change impact mining's rule generation involves four steps of which the adaptive techniques presented herein only affect the first step where they *dynamically* select transactions from which to generate rules. The final three steps are identical to those of previous algorithms. Similar to existing algorithms, if the selection yields an empty list, then no recommendation can be given.

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 with the most recent transaction. There are two motivations for considering this algorithm: first, there is evidence from previous work (Moonen et al., 2018) that a (single) recent relevant transaction can produce very good results. Moreover, 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 straw man to investigate the value brought by the more selective adaptive approaches.

The third adaptive algorithm, *dynamic-block*, aims to capture 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 subsequent adjacent transactions until an irrelevant transaction is encountered. 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 is likely to be prematurely cut off because of the increased likelihood of interjected irrelevant 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}*. Note that we implicitly also consider *dynamic-P_0*, because it uses the same selection constraint

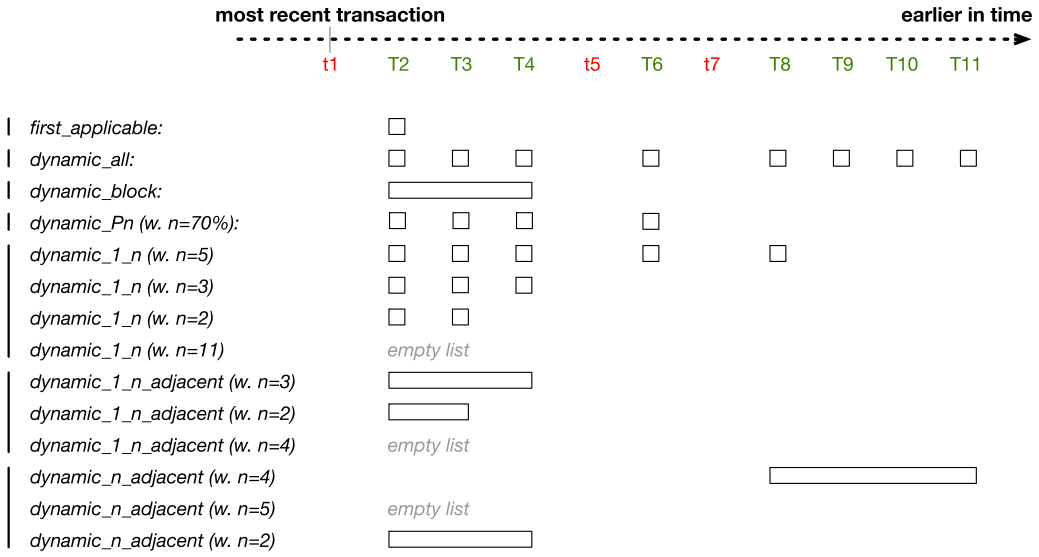

Fig. 1. A visualization of the various adaptive algorithms.

as *dynamic-all*, and *dynamic-P_{100}*, which uses the same selection constraint as *dynamic-block*.

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 decision 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 $n \in \{1, 2, 3, 4, 5, 10, 100, 1000\}$. In addition, note that $dynamic_{1,1}$ is the same as *first-applicable*.

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 $n \in \{1, 2, 3, 4, 5\}$. In addition, note that like $dynamic_{1,1}$, $dynamic_{1,1}adjacent$ is also the same as *first-applicable*.

The seventh family, $dynamic_n adjacent$, aims to increase $dynamic_{1,n}adjacent$'s low applicability while preserving its high accuracy (formalized in Section 5). $Dynamic_n adjacent$ 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 $n \in \{1, 2, 3, 4, 5\}$. Again note that $dynamic_1 adjacent$ is the same as *first-applicable*.

Finally, we consider the application of TARMAQ's transaction selection constraint as an additional step, in conjunction with each of the 24 adaptive algorithms from the first seven families. The resulting algorithms retain only those transactions that are selected by the adaptive algorithm *and* have the largest overlap with the query. Note that the combinations TARMAQ-first-applicable and TARMAQ-dynamic-all are the same as *first-applicable* and TARMAQ, respectively. For the first, TARMAQ will simply pick the single transaction selected by *first-applicable*. For the latter, the set of all relevant transactions that have the largest overlap with the query is exactly the set used by TARMAQ. For the remaining 22 dynamic algorithms, TARMAQ's selection of only those transactions with the largest overlap has two possible effects: on the one hand, the transactions with maximal overlap should provide the strongest evidence, while on the other hand, fewer transactions are considered, which reduces the amount of evidence considered. Our empirical investigation is aimed at evaluating the interaction and relative impact of these two effects.

5. Research questions

To better understand the pros and cons of adaptive algorithms for mining association rules, we investigate the six research questions presented in this section. These research questions make use of the following metrics to quantitatively evaluate algorithm performance.

For a ranked list, such as the mined impact I_q, the standard method for assessing its quality is the list's Average Precision (AP) (Baeza-Yates and Ribeiro-Neto, 1999), while the quality of an algorithm can be assessed by the *mean* Average Precision (MAP) that it achieves over a collection of queries. AP is computed using Precision at k, $P(k)$, which is the *precision* calculated on the first k items in the list (i.e., the fraction of correct artifacts in the top k artifacts). AP is the average of $P(k)$, where k is the rank of each relevant artifact in I_q. In our case, an artifact is relevant if it is found in E_q. More formally,

Definition 2 (Average Precision)**.** Given a query q, its impact I_q, and expected outcome E_q, the average precision, *AP*, of I_q is given by

$$AP(I_q) = \sum_{k=1}^{|I_q|} P(k) * \triangle r(k)$$

where $\triangle r(k)$ is the *change in recall* calculated only on the $k-1^{\text{th}}$ and k^{th} artifacts (i.e., the number of additional correct items predicted compared to the previous rank) (Baeza-Yates and Ribeiro-Neto, 1999).

Table 1 illustrates the computation of *AP*, $P(k)$, and $\triangle r(k)$ given the ranked list $[c, a, f, g, d]$ and the expected outcome $\{c, d, f\}$.

The experiments make use of two different MAP computations: *overall MAP* and *MAP when applicable*. The difference lies in the

Table 1
Calculation of average precision (AP), based on ranked list [*c*, *a*, *f*, *g*, *d*] and expected outcome {*c*, *d*, *f*}.

Rank (k)	Artifact	$P(k)$	$\triangle r(k)$
1	c	1/1	1/3
2	a	1/2	0
3	f	2/3	1/3
4	g	2/4	0
5	d	3/5	1/3

$AP = 1/1 * 1/3 + 1/2 * 0 + 2/3 * 1/3 + 2/4 * 0 + 3/5 * 1/3 \approx 0.75$

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.

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 investigation 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 **TARMAQ As Selector:** What impact does applying TARMAQ as an additional transaction selector have on the adaptive techniques in terms of their (a) viability, (b) applicability, and (c) accuracy, and in addition, (d) how does TARMAQ-as-selector interact with adjacency?

RQ6 **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 rather it is to understand better the interplay between the number of transactions considered and the quality of the recommendation. For example, the first research question would be negatively answered if the adaptive algorithms *never* performed as well as TARMAQ. On the other hand, the more often they do so, the greater the possibility of exploiting them in an ensemble of algorithms.

6. 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.

6.1. 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 2 along with demographics illustrating their diversity. The table shows that the systems vary from medium to large size, with close to 300,000 different files for one system, and nearly 768,000 unique artifacts appearing in another's change history. 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

Table 2
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[a]
CPython	12.05	7725	30,090	Python, C, 16 others
Gecko	1.08	86,650	231,850	C++, C, JavaScript, 34 others
Git	11.02	3753	17,716	C, shell script, Perl, 14 others
Hadoop	6.91	24,607	272,902	Java, XML, 10 others
HTTPD	19.78	10,019	29,216	XML, C, Forth, 19 others
IntelliJ IDEA	2.61	62,692	343,613	Java, Python, XML, 26 others
Liferay Portal	0.87	144,792	767,955	Java, XML, 12 others
Linux Kernel	0.77	26,412	161,022	C, 16 others
LLVM	4.54	25,600	66,604	C++, Assembly, C, 16 others
MediaWiki	11.69	12,252	12,267	PHP, JS, 11 others
MySQL	10.68	42,589	136,925	C++, C, JS, 24 others
PHP	10.82	21,295	53,510	C, PHP, XML, 24 others
Ruby on Rails	11.42	10,631	10,631	Ruby, 6 others
RavenDB	8.59	29,245	139,415	C#, JS, XML, 12 others
Subversion	14.03	6559	46,136	C, Python, C++, 15 others
WebKit	3.33	281,898	397,850	HTML, JS, C++, 24 others
Wine	6.60	8234	126,177	C, 16 others
Cisco	2.43	64,974	251,321	C++, C, C#, Python, others
KM	15.97	35,111	35,111	C++, C, XML, others

[a] languages used by open source systems from http://www.openhub.net.

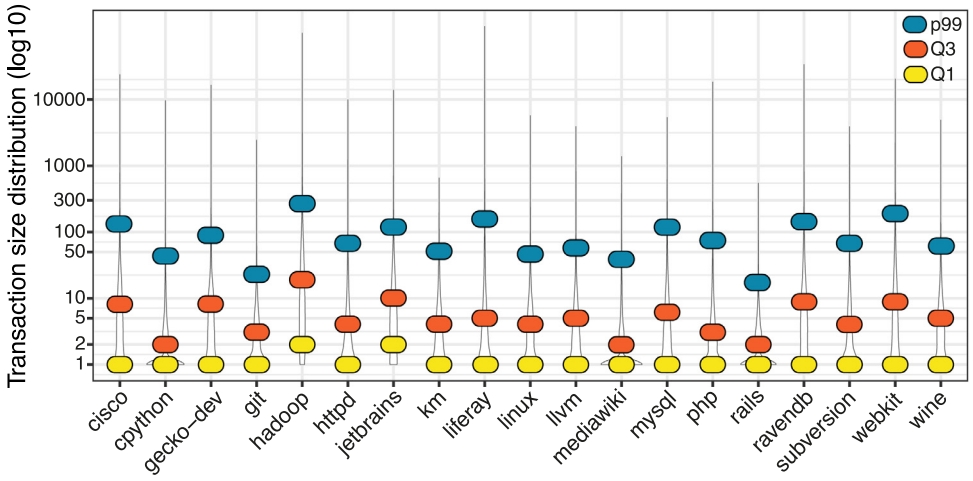

Fig. 2. A plot of the distributions of transactions sizes for each subject system.

in the case of HTTPD, to a little over 10 months in the case of the Linux kernel.

Finally, when experimenting with both ATARI and TARMAQ, we consider *practical fine-grained change histories* (Rolfsnes et al., 2018), which consists of function-level granularity for those source code files that can be parsed using srcML (Collard et al., 2013), and file-level granularity otherwise. We include a *residual* per parsable file to capture the changes to code that is not part of a function (e.g., global variable declarations). Thus *artifacts* in our change history are either a function (or the residual) for parsable source code, or entire files otherwise (e.g., for configuration, build, or documentation artifacts).

6.2. History filtering

Conceptually, a *query q* represents a set of artifacts that a developer changed since their 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. Furthermore, such large transactions lead to a combinatorial explosion in the number of association rules (Agrawal et al., 1993). Fortunately, as shown in the violin plots in Fig. 2, transaction sizes are heavily skewed towards smaller transactions. Unfortunately, as also shown in Fig. 2, there exist outlier transactions containing 10,000 or more artifacts. To handle such outliers, it is common practice to filter the history by removing transactions larger than a certain size (Ying et al., 2004; Zimmermann et al., 2005; Alali, 2008; Kagdi et al., 2013).

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 (indicated by the blue (dark grey) p99 markers in Fig. 2). In most cases, the percentage is well above 99% of the available data.

6.3. Query generation

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. Because a commit is a *set* of artifacts, we do not know the actual order that they were modified or added to the commit. The random selection captures one of many orders. Fortunately, from the standpoint of the empirical evaluation, all orders are equally viable. Finally, to respect the historical time-line, the history used to determine impact based on a query generated from transaction *t*, is composed of only those transactions that are *older* than *t*.

6.4. ATARI implementation

The eight families of algorithms from Section 4 were implemented in the prototype tool ATARI (Adaptive Targeted Association Rule mIning). ATARI was developed in RUBY as a fork of the TARMAQ implementation, which was generously provided to us by its developers (Rolfsnes et al., 2016). Because both tools use the same input- and output formats, it was easy to compare the experimental results.

7. Results

7.1. 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.

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 2, each algorithm produces two outcomes: its applicability and, if applicable, an AP value.

In the comparison, two algorithms *A* and *B 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,

[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 3
Viability data for the 24 adaptive algorithms (algorithm A is always TARMAQ).

Algorithm *B*	A 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	14,626
$dynamic_{1,1000}$	158	95	112	18,584
$dynamic_{1,1}adjacent$	7551	8125	3273	0
$dynamic_{1,2}adjacent$	645	738	493	17,073
$dynamic_{1,3}adjacent$	146	195	171	18,437
$dynamic_{1,4}adjacent$	57	68	75	18,749
$dynamic_{1,5}adjacent$	28	33	33	18,855
$dynamic_{1}$-*adjacent*	7551	8125	3273	0
$dynamic_{2}$-*adjacent*	5850	3453	1898	7748
$dynamic_{3}$-*adjacent*	3371	1757	908	12,913
$dynamic_{4}$-*adjacent*	1629	893	471	15,956
$dynamic_{5}$-*adjacent*	864	522	245	17,318

or if both are applicable and *A* produces a higher AP value. Algorithm *B wins* in the symmetric situation.

Table 3 shows the resulting data (algorithm A is always TARMAQ). 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).

Table 3's fourth column, "*B* wins", shows that each of the adaptive algorithms has at least a handful 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 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, the answer to RQ1 is that the adaptive analysis is not completely subsumed by TARMAQ, and consequently, the adaptive algorithms are viable.

7.2. 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 comparison uses the applicability data obtained by running the 24 adaptive algorithms and TARMAQ on the 20,900 queries described in Section 6.3.

The range of applicabilities is seen on the *x*-axis of Fig. 3. 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 equal TARMAQ's applicability. Moving to the left, the adaptive techniques have progressively lower applicability. As shown in Table 4a, 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 the constraints grow more stringent, the applicability decreases. Two constraints were considered, the value of *n* and the requirement that the selected transactions be adjacent. There is a notable drop-off in applicability as the value of *n* increases and with the requirement of adjacency. This drop-off is considerably sharper when the two are combined. The examination of RQ4 takes a deeper look at adjacency's impact. Finally, compared to TARMAQ, 40% of the adaptive algorithms match TARMAQ's applicability while the remainder grows progressively less applicable as *n* increases.

7.3. 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 6.3. We consider two views of the data: *overall MAP*, which is *harsh* because it assigns an AP of zero when an algorithm is not applicable, and *MAP when applicable*, which is *kind* because it ignores queries to which an algorithm is not applicable.

Table 4b presents the results of Tukey's HSD for the *overall MAP* values. This statistical test identifies potential differences between the adaptive algorithms and TARMAQ. The results find TARMAQ superior to all of the adaptive algorithms as it is the only algorithm in the top group (labeled *a*). 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 that they make use of, it is interesting to see how close some of them come. For example, using only the first applicable transaction lowers MAP by only 11%.

Turning to *MAP when applicable*, the *y*-axis of Fig. 3 shows the range of values for the adaptive algorithms and TARMAQ. Statistically, as seen in Table 4c, a difference only exists between the top three and bottom five algorithms. This large overlap in accuracy suggests that adaptive algorithms might be selected over TARMAQ in cases where only limited history is available, or where using less history reduces response time.

In summary, for RQ3, as expected, none of the adaptive algorithms is a clear replacement for TARMAQ. However, given how well some of them perform when applicable, and how little of the history they need, hybrid approaches may be able to outperform TARMAQ while keeping its high applicability. This concept is considered in greater detail in Section 8.

7.4. RQ4: Adjacency's impact

RQ4 investigates the impact on both applicability and accuracy of requiring that relevant transactions be *adjacent*. Adjacency of relevant transactions suggests that a developer is working on a single issue. Thus, the information gleaned from such transactions should identify key relations between software artifacts. Two of the families, $dynamic_{1,n}adjacent$ and $dynamic_{n}adjacent$, require adjacent transactions. Recall the difference between these two families: $dynamic_{1,n}adjacent$ 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.

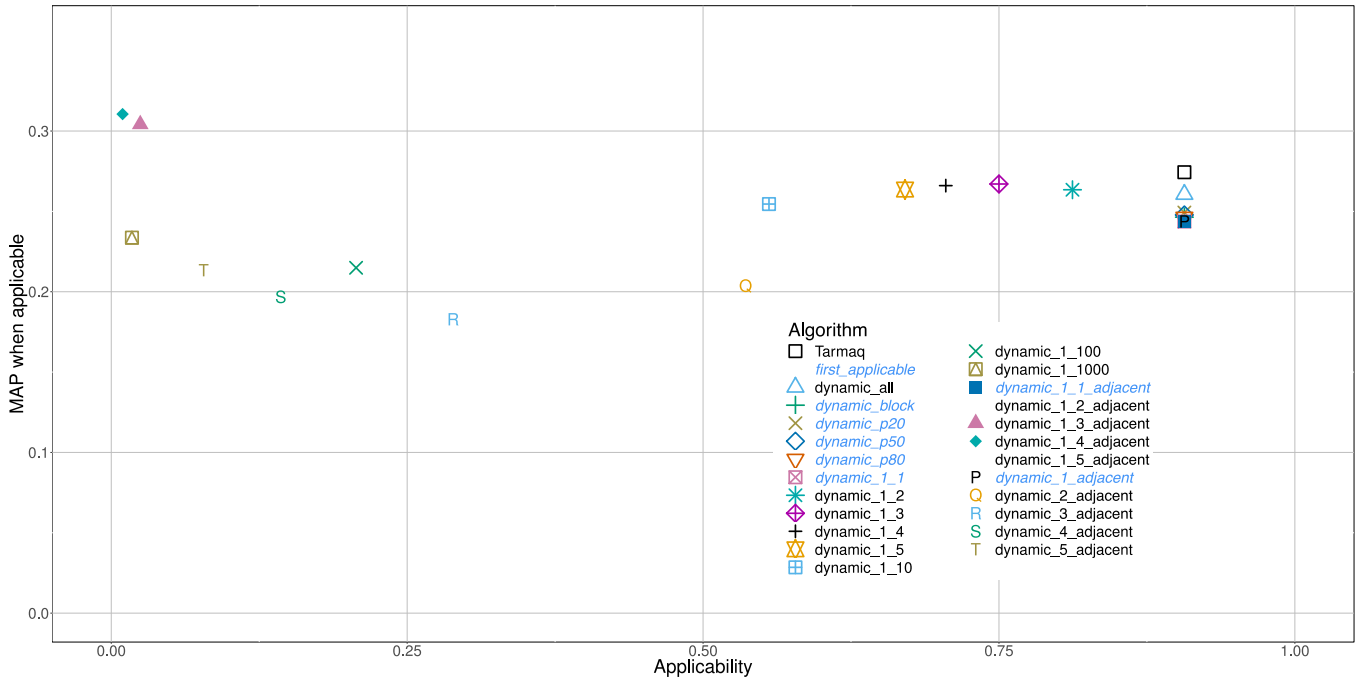

Fig. 3. A visualization of MAP and applicability for the 24 adaptive algorithms and TARMAQ. The legend indicates in italics and color the eight algorithms whose symbols overlap in the graph. (A color version of this figure can be found in the web version of this article.)

Table 4
Tukey's HSD analysis for Applicability, Overall MAP and MAP when Applicable.

(a) Applicability			(b) Overall MAP			(c) MAP when applicable		
Algorithm	Appl.	Group	Algorithm	MAP	Group	Algorithm	MAP	Group
$dynamic_{1,1}$	0.9066	*a*	TARMAQ	0.2487	*a*	$dynamic_{1,4}adj$	0.3105	*a*
$dynamic_{1,1}adj$	0.9066	*a*	*dynamic-all*	0.2362	*b*	$dynamic_{1,2}adj$	0.3084	*a*
$dynamic_1$-*adj*	0.9066	*a*	*dynamic-*P_{20}	0.2260	*c*	$dynamic_{1,3}adj$	0.3042	*a*
dynamic-all	0.9066	*a*	*dynamic-*P_{50}	0.2246	*c*	$dynamic_{1,5}adj$	0.2848	*ab*
dynamic-block	0.9066	*a*	*dynamic-block*	0.2233	*cd*	TARMAQ	0.2744	*ab*
dynamic-P_{20}	0.9066	*a*	*dynamic-*P_{80}	0.2233	*cd*	$dynamic_{1,3}$	0.2670	*ab*
dynamic-P_{50}	0.9066	*a*	$dynamic_{1,1}$	0.2209	*cd*	$dynamic_{1,4}$	0.2660	*ab*
dynamic-P_{80}	0.9066	*a*	$dynamic_{1,1}adj$	0.2209	*cd*	$dynamic_{1,5}$	0.2636	*ab*
first-applicable	0.9066	*a*	$dynamic_1$-*adj*	0.2209	*cd*	$dynamic_{1,2}$	0.2635	*ab*
TARMAQ	0.9066	*a*	*first-applicable*	0.2209	*cd*	*dynamic-all*	0.2606	*ab*
$dynamic_{1,2}$	0.8121	*b*	$dynamic_{1,2}$	0.2139	*d*	$dynamic_{1,10}$	0.2546	*ab*
$dynamic_{1,3}$	0.7500	*c*	$dynamic_{1,3}$	0.2002	*e*	*dynamic-*P_{20}	0.2493	*ab*
$dynamic_{1,4}$	0.7050	*d*	$dynamic_{1,4}$	0.1875	*f*	*dynamic-*P_{50}	0.2478	*ab*
$dynamic_{1,5}$	0.6706	*e*	$dynamic_{1,5}$	0.1767	*g*	*dynamic-block*	0.2464	*ab*
$dynamic_{1,10}$	0.5556	*f*	$dynamic_{1,10}$	0.1414	*h*	*dynamic-*P_{80}	0.2464	*ab*
$dynamic_2$-*adj*	0.5359	*g*	$dynamic_2$-*adj*	0.1091	*i*	$dynamic_{1,1}$	0.2437	*ab*
$dynamic_3$-*adj*	0.2888	*h*	$dynamic_3$-*adj*	0.0527	*j*	$dynamic_{1,1}adj$	0.2437	*ab*
$dynamic_{1,100}$	0.2068	*i*	$dynamic_{1,100}$	0.0444	*j*	$dynamic_1adj$	0.2437	*ab*
$dynamic_4$-*adj*	0.1432	*j*	$dynamic_4$-*adj*	0.0281	*k*	*first-applicable*	0.2437	*ab*
$dynamic_{1,2}adj$	0.0897	*k*	$dynamic_{1,2}adj$	0.0276	*k*	$dynamic_{1,1000}$	0.2337	*ab*
$dynamic_5$-*adj*	0.0780	*l*	$dynamic_5$-*adj*	0.0166	*l*	$dynamic_{1,100}$	0.2149	*b*
$dynamic_{1,3}adj$	0.0244	*m*	$dynamic_{1,3}adj$	0.0074	*lm*	$dynamic_5adj$	0.2133	*b*
$dynamic_{1,1000}$	0.0174	*mn*	$dynamic_{1,1000}$	0.0040	*m*	$dynamic_2adj$	0.2036	*b*
$dynamic_{1,4}adj$	0.0095	*no*	$dynamic_{1,4}adj$	0.0029	*m*	$dynamic_4adj$	0.1968	*b*
$dynamic_{1,5}adj$	0.0044	*o*	$dynamic_{1,5}adj$	0.0012	*m*	$dynamic_3adj$	0.1828	*b*

NB: note that the *adjacent* adaptive algorithms are abbreviated to *adj* in these tables to save space.

The analysis considers both inter-family and intra-family trends. Beginning with applicability, Fig. 4 summarizes the data and also includes $dynamic_{1,n}$ for comparison. 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 $dynamic_nadjacent$).

As evident in the figure, the applicability of $dynamic_{1,n}$ has the least drastic drop-off. In fact, as far out as $n = 1000$ it still manages an applicability of 2% (i.e., 2% of the queries have at least 1000 relevant transactions). At the other end of the spectrum, $dynamic_{1,n}adjacent$ has the fastest reduction in applicability. For example, its drop in applicability from $n = 1$ to $n = 2$ is 82%. Finally, $dynamic_nadjacent$'s applicability trend lies in between the other two.

Overall, 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 the more rapid falloff in applicability, which indicates that large runs of relevant transactions are rare in the change histories. Unsurprisingly, the additional requirement that the run of adjacent transactions begins with the first relevant

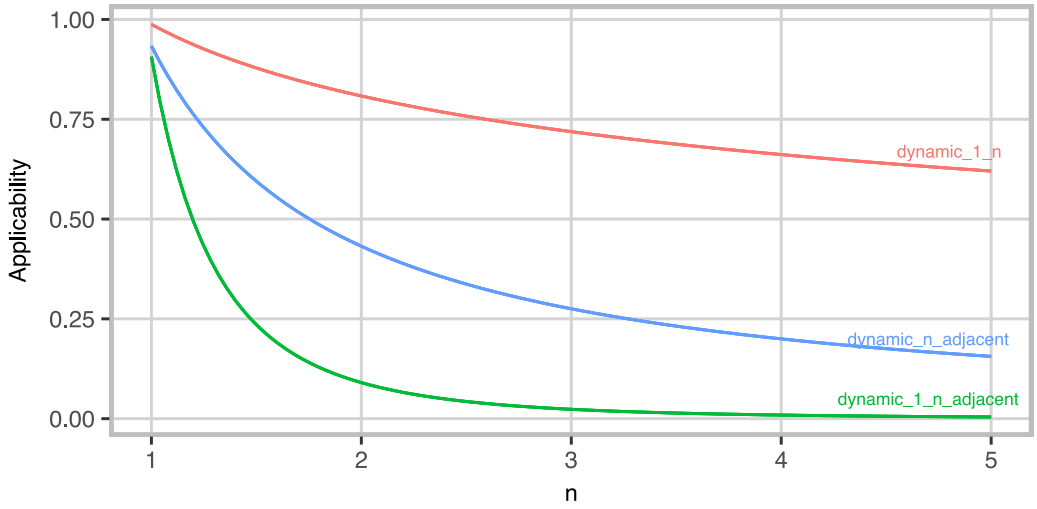

Fig. 4. Applicability as a function of *n*.

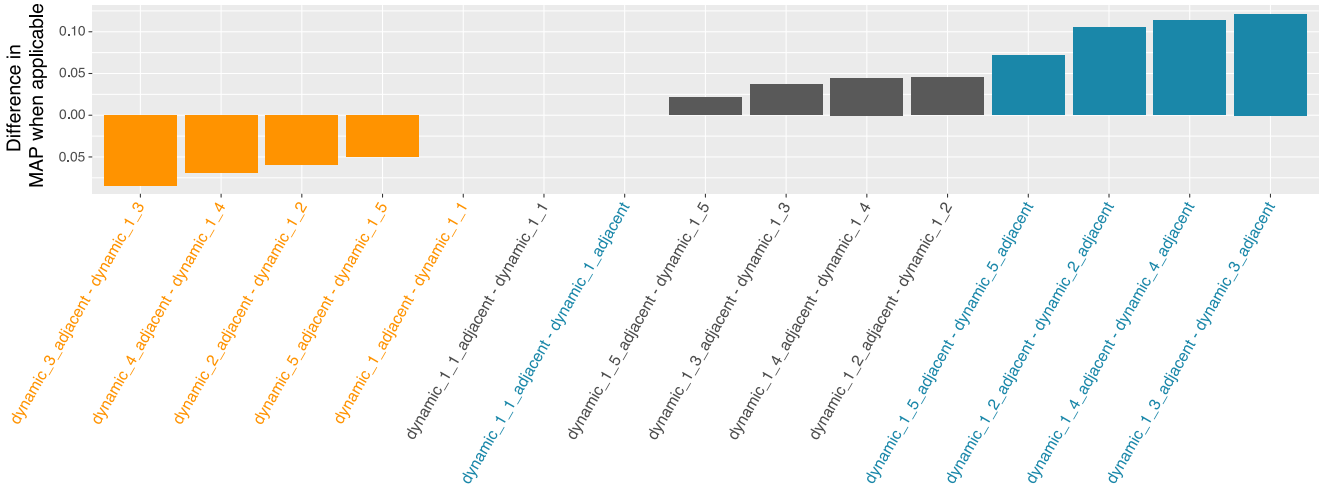

Fig. 5. Adjacency's impact on *MAP when applicable* (the colors refer to the three head-to-head comparisons discussed in the text). (A color version of this figure can be found in the web version of this article.)

transaction leads to $dynamic_{1,n}adjacent$ showing the most dramatic decrease.

For the accuracy comparison, it is useful to bound the values of n considered. To do so, we consider the value of n at which an algorithm's applicability falls below 1%. $Dynamic_{1,n}adjacent$ falls below 1% for values of n greater than four, while $dynamic_nadjacent$ does so for values greater than nine. For completeness, $dynamic_{1,n}$ does not fall below the threshold until n exceeds 2500. Because of limited applicability, we focus the accuracy investigation on values of n ranging from one to five.

Fig. 5 illustrates adjacency's impact on accuracy. The accuracy analysis involves three head-to-head comparisons: first we compare $dynamic_{1,n}$ with $dynamic_nadjacent$ (shown in orange (light grey) in the figure), then $dynamic_{1,n}$ with $dynamic_{1,n}adjacent$ (shown in black), and finally $dynamic_{1,n}adjacent$ with $dynamic_nadjacent$ (shown in teal (dark grey)). For comparisons involving $dynamic_{1,n}$ (the orange and black bars), a positive difference occurs when requiring adjacency leads to a higher MAP value, while a negative difference occurs when requiring adjacency leads to a lower MAP value. For example, the first bar shows that $dynamic_3adjacent$ underperforms $dynamic_{1,3}$ (by about eight and a half percentage points). Finally, note that for $n = 1$, $dynamic_{1,n}$, $dynamic_{1,n}adjacent$, and $dynamic_nadjacent$ select the same single transaction, thus all three differences are zero.

Consider first the comparison of $dynamic_{1,n}$ and $dynamic_nadjacent$ (shown in orange in Fig. 5). Recall that $dynamic_{1,n}$ uses the most recent n transactions. In contrast, because it is not required to start at the first relevant transaction, $dynamic_nadjacent$ often makes use of older transactions. It is helpful to note that whenever $dynamic_nadjacent$ is applicable, then so is $dynamic_{1,n}$, although the two may use different transactions. As shown by the orange bars in the chart, adjacency always has a negative impact on accuracy for these comparisons. From $n = 1$ to $n = 3$ the cost of adjacency grows (the orange bars grow taller) while for $n = 3$ to $n = 5$ its negative effect diminishes (the orange bar grow shorter). There are likely two effects here: first, as n increases, $dynamic_nadjacent$ searches further back in the history to find n adjacent transactions. Greater age tends to have a negative impact on accuracy. However, the decreasing negative difference in the value of *MAP when applicable* as n goes from 3 to 5 suggests that a larger number of adjacent transactions can counter the negative effects of increased age.

Statistically, we compare the two families side-by-side for every value of n other than 1, using the Wilcoxon rank-sum test (also known as the Mann-Whitney U test). For example, for $n = 2$, we apply the test to the data collected for $dynamic_{1,2}$ and $dynamic_2adjacent$. Although our experiment generates paired data for which the statistically stronger Wilcoxon signed-rank tests can

be used, it is possible that two dynamic algorithms select the same transactions and give the exact same recommendation, making their difference zero. Such ties cannot be handled by the Wilcoxon signed-rank test and would need to be removed. However, in certain cases, such as the comparison of $dynamic_{1,2}$ and $dynamic_2 adjacent$, a considerable part of the data would need to be removed. Because we prefer to use all our data points than base the comparison on varying sets of data with zeroes removed, we choose the slightly weaker Wilcoxon rank-sum test that does not treat the data as being paired, and can handle zero differences. With a p-value < 0.0001, in each case the negative impact of requiring adjacency is statistically significant for each value of n.

Considering next the comparison of $dynamic_{1,n}$ and $dynamic_{1,n} adjacent$ (shown in black in Fig. 5). In this case, both families include the first relevant transaction. Furthermore, if $dynamic_{1,n} adjacent$ is applicable then so is $dynamic_{1,n}$ and the two will use the exact same transactions. However, when only $dynamic_{1,n}$ is applicable, it will necessarily make use of older transactions. Said another way, on average $dynamic_{1,n} adjacent$ uses more recent transactions.

As shown by the black bars in the chart, adjacency has a positive impact on accuracy for this group of comparisons. The impact fluctuates as the value of n increases, with an indication that by $n = 5$, the initial accuracy gap is disappearing. The implication here is that initially age dominates, but at some point (e.g., $n \geq 5$), the volume of data starts to counteract the cost of using older transactions.

Statistically, when comparing $dynamic_{1,n}$ and $dynamic_{1,n} adjacent$ for $n = 2$ *to* 5 using the Wilcoxon rank-sum test, the resulting p-values are < 0.0001, 0.00033, 0.02244, and 0.467. In other words the difference is statistically significant for $n = 2$, 3, *and* 4, although the difference for 4 is not large. It is worth noting that for $n = 5$ there are only 94 applicable queries for $dynamic_{1,5} adjacent$, which is less than 0.5% of the 20 900 data points considered. This low number and the smaller difference in *MAP when applicable* limits the statistical test's ability to establish the significance of any difference. In summary, for $dynamic_{1,n}$, applying adjacency yields better MAP values at the expense of applicability.

The final comparison considers $dynamic_{1,n} adjacent$ and $dynamic_n adjacent$ (shown in teal in Fig. 5). Since these two families both use adjacent transactions, the only difference between them is the transaction's age. In this case whenever $dynamic_{1,n} adjacent$ is applicable then so is $dynamic_n adjacent$ and furthermore the two use the same transactions. However, if the run of n adjacent transaction is not associated with the first relevant transaction then only $dynamic_n adjacent$ is applicable.

In contrast to the previous two comparisons, when these two differ, it is only because of the transaction age. It is quite clear from the graph that transaction age matters, likely because older transactions contribute less relevant evidence to a recommendation. Head-to-head statistical comparisons for $n = 2$ *to* 5 each yield a strong statistically significant difference (p-value < 0.0001). Thus dynamic$_n$adjacent has greater applicability but lower MAP than dynamic$_{1,n}$adjacent.

In summary, for RQ4 adjacency has a significant cost in terms of applicability. This is not unexpected. In exchange, it notably improves accuracy, which, however, is dampened by the age of the transactions involved.

7.5. RQ5: TARMAQ as selector

RQ5 considers the impact of applying TARMAQ as an additional selection constraint to the transactions selected by an adaptive algorithm. In general, the most useful recommendations are not cluttered by irrelevant software artifacts. Such recommendations achieve higher AP values and therefore are of greater utility to engineers. Removing transactions involving artifacts that have a low likelihood of relevance is one of TARMAQ's goals. The remainder of this subsection replicates the consideration of the proceeding four research questions while incorporating TARMAQ-as-selector.

Table 5
Viability data for the 24 adaptive algorithms that use TARMAQ-as-selector (algorithm A is always TARMAQ).

Algorithm *B*	A wins	AP tie	B wins	B not applicable
TARMAQ-*first-applicable*	7551	8125	3273	0
TARMAQ-*dynamic-all*	1335	15,736	1878	0
TARMAQ-*dynamic-block*	7340	8309	3300	0
TARMAQ-*dynamic-*P_{20}	7020	8578	3351	0
TARMAQ-*dynamic-*P_{50}	7235	8389	3325	0
TARMAQ-*dynamic-*P_{80}	7346	8287	3316	0
TARMAQ-$dynamic_{1,1}$	7551	8125	3273	0
TARMAQ-$dynamic_{1,2}$	5730	7593	3650	1976
TARMAQ-$dynamic_{1,3}$	4735	7219	3721	3274
TARMAQ-$dynamic_{1,4}$	4137	6941	3658	4213
TARMAQ-$dynamic_{1,5}$	3771	6708	3537	4933
TARMAQ-$dynamic_{1,10}$	2598	5923	3093	7335
TARMAQ-$dynamic_{1,100}$	518	2795	1010	14,626
TARMAQ-$dynamic_{1,1000}$	9	325	31	18,584
TARMAQ-$dynamic_{1,1} adjacent$	7551	8125	3273	0
TARMAQ-$dynamic_{1,2} adjacent$	602	872	402	17,073
TARMAQ-$dynamic_{1,3} adjacent$	121	267	124	18,437
TARMAQ-$dynamic_{1,4} adjacent$	46	96	58	18,749
TARMAQ-$dynamic_{1,5} adjacent$	19	48	27	18,855
TARMAQ-$dynamic_1$-*adjacent*	7551	8125	3273	0
TARMAQ-$dynamic_2$-*adjacent*	5767	3712	1722	7748
TARMAQ-$dynamic_3$-*adjacent*	3306	1946	784	12,913
TARMAQ-$dynamic_4$-*adjacent*	1595	985	413	15,956
TARMAQ-$dynamic_5$-*adjacent*	846	572	213	17,318

(RQ5a) Viability: To consider the viability of TARMAQ as additional selector, we do a similar analysis as with RQ1. Table 5 shows the results of executing TARMAQ and each of the 24 adaptive algorithms with TARMAQ-as-selector on the 1100 randomly selected queries from each of the 19 systems. As before, no algorithm is ever applicable when TARMAQ is not, and for the same 1951 of the 20 900 queries, none of the algorithms were able to make a recommendation.

When we compare the third columns of Tables 5 and 3, we see an increase in ties when using TARMAQ-as-selector. However, the fourth column in Table 5 still shows that each of the adaptive algorithms using TARMAQ-as-selector has queries for which it outperforms TARMAQ.

Overall, this leads us to the conclusion that the adaptive algorithms that use TARMAQ as additional selector are not completely subsumed by TARMAQ's results, and consequently, are viable.

(RQ5b) Applicability: The addition of TARMAQ as a selector has no impact on the applicability of the adaptive algorithms because the selection always leaves a non-empty set. Thus, this step will never remove all the transactions, so if an algorithm was applicable to a given query before, it will remain applicable to that query. Visually the unchanged applicability can be observed by comparing the x-axis position of each algorithm in Fig. 3 against its position in Fig. 6, which repeats Fig. 3 when including TARMAQ as additional selector.

(RQ5c) Accuracy: The analysis considers the accuracy data obtained by running the 24 adaptive algorithms both with and without TARMAQ as additional transaction selector (i.e., 48 algorithms in total). The algorithms are run on the queries described in Section 6.3. Visually the impact of applying TARMAQ-as-selector can be observed by comparing the vertical position (the MAP value) of each algorithm in Fig. 3 against that in Fig. 6. This comparison clearly shows that for several of the adaptive algorithms when using TARMAQ-as-selector yield higher MAP values than TARMAQ (although, as before, they do so at the expense of applicability).

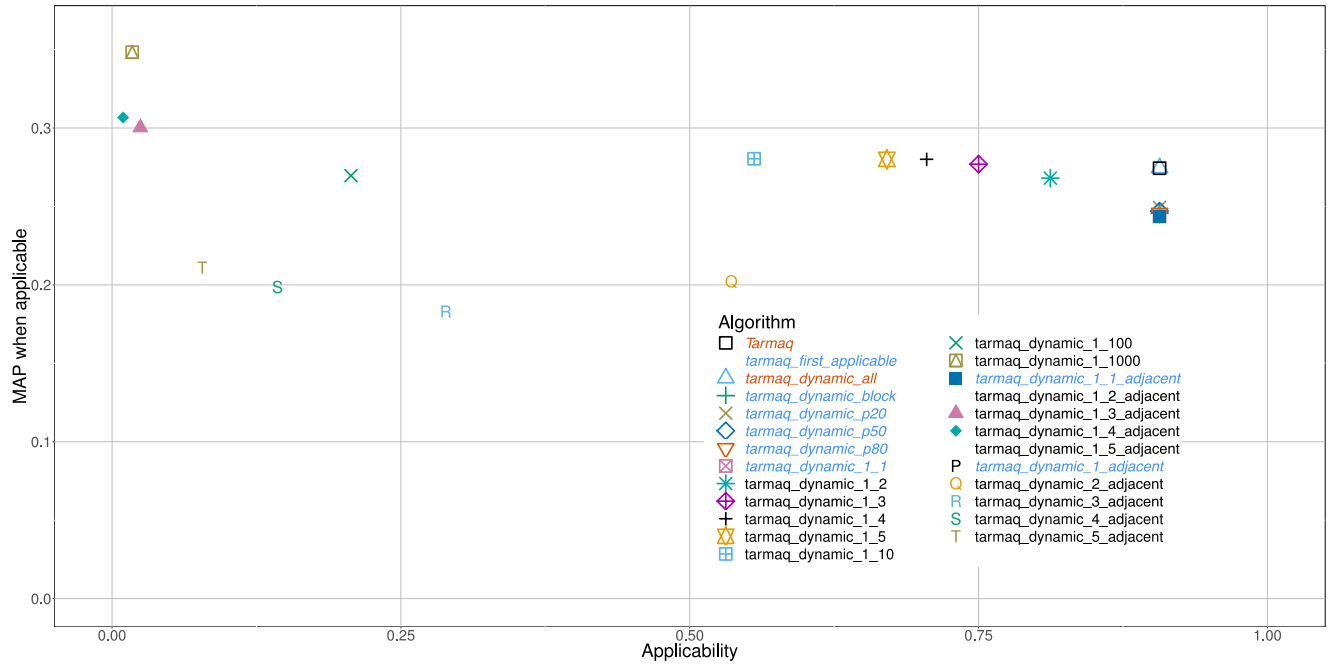

Fig. 6. A visualization of MAP and applicability for Tarmaq and the 24 adaptive algorithms with Tarmaq -as-selector. The legend indicates in italics and color the algorithms whose symbols overlap in the plot. (A color version of this figure can be found in the web version of this article.)

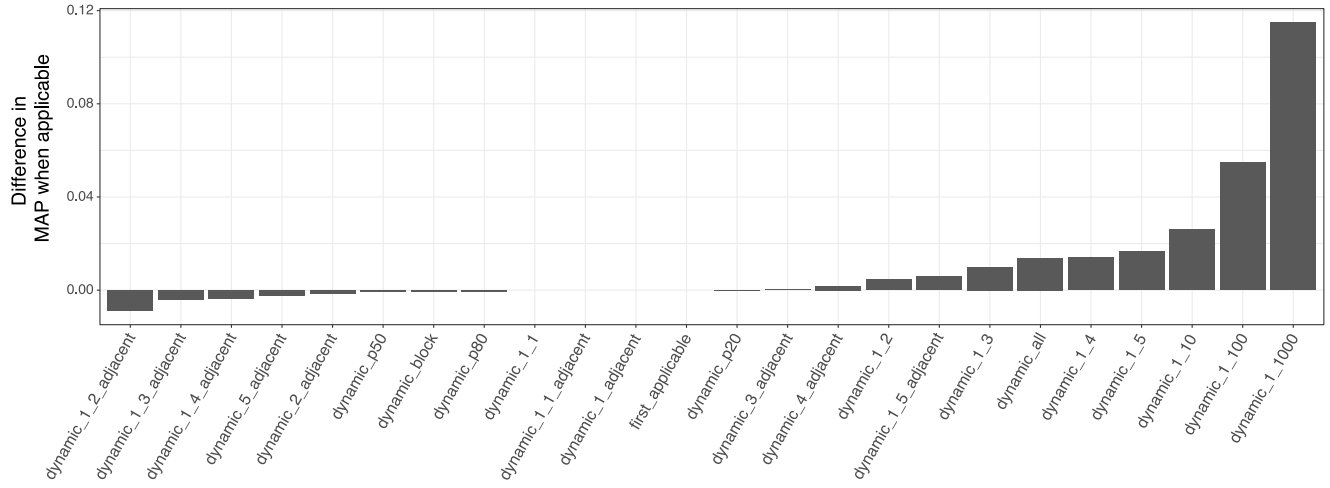

Fig. 7. Impact on *MAP when applicable* of using Tarmaq as additional selection constraint for each of the 24 adaptive techniques.

Similar to Figs. 5 and 7 shows the *difference* in *MAP when applicable* that is caused by applying Tarmaq-as-selector for each of the 24 adaptive algorithms, shown on the *x*-axis. A positive difference indicates that adding Tarmaq-as-selector improved accuracy, while a negative difference indicates that the addition lowered accuracy. Overall, Tarmaq-as-selector has a positive effect on the accuracy of the adaptive algorithms. However, the improvement is not universal. The remainder of the discussion of RQ5c takes a deeper look at the impact that Tarmaq-as-selector has on the three solitary adaptive algorithms before considering family trends.

First, recall that *dynamic-all* considers the largest possible number of transactions as it considers all the relevant transactions. As seen in Fig. 7, the MAP value for *dynamic-all* improves when Tarmaq-as-selector is applied. Here Tarmaq-*dynamic-all* is identical to the original Tarmaq algorithm and thus this result is consistent with prior work (Rolfsnes et al., 2016).

At the other end of the spectrum, *first-applicable* selects only a single transaction and thus using Tarmaq-as-selector has no effect on its accuracy. The same is true of $dynamic_{1,1}$, $dynamic_{1,1}$*adjacent*, and $dynamic_1$*adjacent*.

Table 6
Basic statistics for the number of transactions used by *dynamic-block*.

Q1	Median	Mean	Q3	Max.
1.0	1.0	1.044	1.0	15.0

Despite *dynamic-all* and *first-applicable* bookending the number of transactions selected, it is interesting to note that, as explored below, they do not bookend the impact of applying Tarmaq as a selector. *Dynamic-block*, the final family-of-one, illustrates this. While the difference is small, Tarmaq-as-selector negatively impacts the accuracy of *dynamic-block*. To a developer, the small difference in algorithm accuracy is likely insignificant. However, it is useful to understand its cause.

Table 6 shows some basic statistics regarding the number of transactions selected by *dynamic-block*. To begin with, note that 90% of the time, a single transaction is selected. In these cases, TARMAQ-as-selector has no effect and hence there is no accuracy difference. The selector is thus having a negative impact on the accuracy of the remaining sizes. In other words, it is discarding useful information. Broken down by the number of transactions, it is not until *dynamic-block* selects nine transactions that the selector has a positive impact on the MAP value. This suggests that for its application to be advantageous, a larger number of transactions is required. Further support for this observation is seen at the far right of Fig. 7 and considered below.

Turning to the families of more than one algorithm, we next consider how the accuracy of *dynamic-*P_n varies with n. For completeness, recall that *dynamic-*P_0 uses the same selection as *dynamic-all*, and *dynamic-*P_{100} uses the same selection as *dynamic-block*. Big picture, lower percentages yield positive MAP differences while higher percentages yield negative MAP differences. Because as n increases the number of transactions selected by *dynamic-*P_n tends to decrease, the data for *dynamic-*P_n reinforces the notion that TARMAQ-as-selector is best applied when a larger number of transactions is involved.

Next, TARMAQ-as-selector has a positive effect on $dynamic_{1,n}$. Furthermore, as the value of n increases, the effect increases. (A best-fit curve suggests the relationship between n and the effect of the selector is polynomial with an R^2 of 0.97.) Similar to *dynamic-*P_n, this data supports that TARMAQ-as-selector works better when provided a larger number of transactions. In summary for RQ5c, the interaction between TARMAQ-as-selector and the dynamic selection performed by the adaptive algorithms is complex. However, despite this complexity, it is clear that applying TARMAQ-as-selector requires a certain minimum amount of data to be effective.

(RQ5d) Interaction with Adjacency: Included in Fig. 7 are the differences in *MAP when applicable* caused by TARMAQ-as-selector on the two families of adaptive algorithms that require adjacency, $dynamic_n adjacent$ and $dynamic_{1,n} adjacent$. In the analysis, we ignore the algorithms with $n = 1$ where TARMAQ-as-selector has no effect.

First consider the family $dynamic_{1,n} adjacent$ where the selector has an initial negative impact, but grows positive by $n = 5$. This again implies that there exists good information in the small sets of adjacent transactions that TARMAQ-as-selector discards. For example, when $n = 2$ the two transactions selected either have the same overlap, in which case both are retained and accuracy is unaffected, or the selector discards one of the two. It is not hard to imagine two transactions with similar (large) overlaps where the slightly smaller overlap provides useful information. Supporting this general pattern, by the time $n = 5$ TARMAQ-$dynamic_{1,5} adjacent$ outperforms $dynamic_{1,5} adjacent$. This again suggests that when sufficient transactions exist, constraining the selection brings value.

For the second family, $dynamic_n adjacent$, we see a varying effect when applying TARMAQ-as-selector. For larger values of n, $dynamic_n adjacent$ tends to search further back in the history. Greater transaction age has a negative effect on accuracy (likely because such transactions contain less relevant information). There appears to be an interplay between the number of transactions n, the adjacency requirement, and the transaction age, which causes the selector to have a varying effect on algorithm accuracy. For example, when n is small (i.e., $n = 1, 2$), TARMAQ-as-selector clearly throws away useful information. However, for larger values of n (i.e., $n = 3, 4$) the number of transactions and value of adjacency complement the effect of TARMAQ-as-selector. For even larger values of n, the negative effect of age overpowers the effect of adjacency and of TARMAQ-as-selector.

Statistically, Table 7 shows the result of applying Tukey's HSD to the MAP differences for $dynamic_{1,n}$ and *dynamic-*P_n. Here "MAP difference" is the difference in MAP value with and without applying TARMAQ as additional selector. These two families were chosen because they best illustrate the impact of TARMAQ -as-selector. Both tables support the conclusion that TARMAQ-as-selector is more beneficial when a larger number of transactions is available. For $dynamic_{1,n}$, as n increases TARMAQ-as-selector is statistically shown to provide an ever-increasing improvement in accuracy (each algorithm is in its own group except for $n = 4, 5$).

Table 7
Tukey's HSD for TARMAQ selector MAP Difference.

Algorithm	MAP Difference	Group
$dynamic_{1,1000}$	0.1146	*a*
$dynamic_{1,100}$	0.0547	*b*
$dynamic_{1,10}$	0.0258	*c*
$dynamic_{1,5}$	0.0164	*d*
$dynamic_{1,4}$	0.0140	*d*
$dynamic_{1,3}$	0.0099	*e*
$dynamic_{1,2}$	0.0046	*f*
$dynamic_{1,1}$	0.0000	*g*
Algorithm	**MAP Difference**	**Group**
dynamic-P_0*	0.1385	*a*
dynamic-P_{20}	0.0008	*b*
dynamic-P_{100}*	-0.0062	*b*
dynamic-P_{80}	-0.0065	*b*
dynamic-P_{50}	-0.0082	*b*

* recall that *dynamic-*P_0 is *dynamic-all*, and *dynamic-*P_{100} is *dynamic-block*.

Table 8
Basic statistics for the number of transactions used for *dynamic-*P_n.

n	Min.	Q1	Median	Mean	Q	Max.
0	1.0	4.0	19.0	105.6	86.0	4281.0
20	1.0	1.0	1.0	1.5	1.0	42.0
50	1.0	1.0	1.0	1.3	1.0	37.0
80	1.0	1.0	1.0	1.2	1.0	30.0
100	1.0	1.0	1.0	1.2	1.0	15.0

For *dynamic-*P_n, this trend exists as well, but it is less pronounced. Specifically, *dynamic-*P_0 (i.e., dynamic-all) shows a significantly greater difference in MAP value than the other four. That there is no statistical difference between the remaining four family members, is supported by the statistics related to the number of transactions shown in Table 8, where most blocks outside of those selected by *dynamic-*P_0 have size 1. This again supports the notion that TARMAQ-as-selector is best applied to a larger number of transactions.

Finally, Fig. 8 compares the fraction of the transactions used by each algorithm against the resulting MAP value for TARMAQ and all the adaptive algorithms. As can be seen in the figure, TARMAQ and *dynamic-all* use the same number of transactions. Sandwiched in between the two *first-applicable* uses a vanishingly small fraction of the data. At the other end of the spectrum, $dynamic_{1,1000}$ unsurprisingly has the highest average, because it *requires* by definition 1000 relevant transactions to be applicable. The figure also helps visualize just how little of the data the *dynamic-*P_n family uses and how similar this usage is, which explains the similarity in the family member's performance. Next to these the slow increase for $dynamic_{1,n}$ as n increases is evident. Algebraically, the values for $dynamic_{1,n}$, $dynamic_n adjacent$ and $dynamic_{1,n} adjacent$ are always n. Finally, considering the MAP values makes it visually apparent that the amount of history used and the resulting MAP values are largely uncorrelated.

While not shown a related analysis considers the number of (non-relevant) transactions that have to be considered while se-

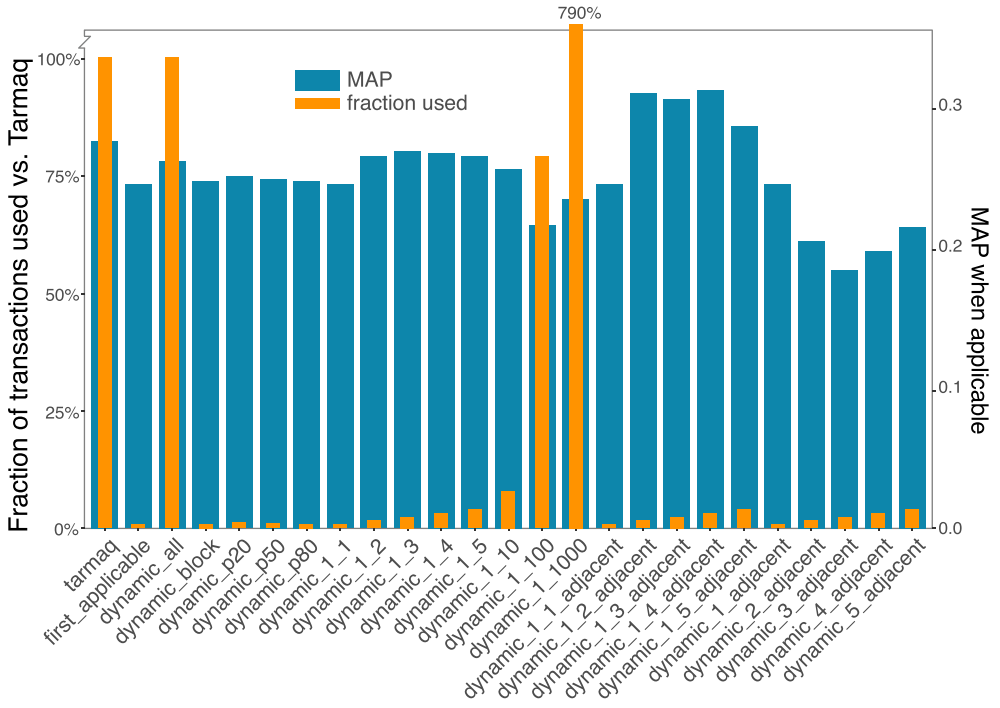

Fig. 8. Fraction of transactions used by the adaptive algorithms with respect to TARMAQ, together with the corresponding MAP that is achieved. Note that $dynamic_{1,1000}$ was truncated to keep a legible scale, its value is shown at the top of the plot.

lecting the relevant used by a given algorithm. For example, one effect found with $dynamic_{1,n}$ is that as n gets larger, a greater number of queries fail to find n relevant transactions. This turns out to be expensive because it requires considering the entire history (to learn that the query is not applicable) and thus the fraction of the history considered grown ever higher.

Algebraically, it is interesting to note that in terms of the transactions considered $dynamic_{1,n}adjacent$ has a significant advantage: the value for $dynamic_{1,n}adjacent$ is always no more than that of $dynamic_{1,n}$, and often much less. We will use the $n = 2$ algorithms to illustrate why this is true (the explanation for other values of n is similar). Consider the following three cases: (1) $dynamic_{1,2}$ is not applicable, or it is applicable, and either (2) the transaction adjacent to the first applicable transaction is also applicable, or (3) it is not. In Cases (1) and (2) both $dynamic_{1,2}$ and $dynamic_{1,2}adjacent$ consider the exact same transactions. However, in Case (3), $dynamic_{1,2}adjacent$ need only consider the adjacent transaction, while $dynamic_{1,n}$ will inspect the remaining history in the hope of finding a relevant transaction. Thus, $dynamic_{1,2}adjacent$ never considers more of the history then $dynamic_{1,2}$.

In summary for RQ5, applying TARMAQ as additional selector to the transactions selected by an adaptive algorithm is viable, does not affect the applicability, and tends to improve the accuracy of the recommendations. However, this result is not universal. TARMAQ-as-selector works best when provided with a sufficient number of transactions. It is interesting to note that this is not a strictly monotonically increasing relation as the algorithm selecting the largest number of transactions, *dynamic-all* does not produce the largest improvement then TARMAQ is applied as additional transaction selector.

7.6. RQ6: Practical implications

By design, adaptive algorithms make use of significantly fewer transactions from the change history than TARMAQ. Moreover, we learned from Fig. 3 that some adaptive algorithms, such as $dynamic_{1,2}$ and $dynamic\text{-}P_{20}$, have equal or only slightly lower applicability than TARMAQ, while maintaining respectable MAP values. Fig. 9 shows the time it takes to make a recommendation when all relevant transactions are extracted and available for use (to enable a fair comparison between "raw" recommendation speeds of TARMAQ vs. the adaptive algorithms). This Figure illustrates that, without significant loss of applicability and accuracy, it is possible to achieve a dramatic reduction in the time it takes to make a recommendation. The time reduction directly comes from the fact that adaptive algorithms consider dramatically fewer transactions (e.g., $dynamic_{1,2}$ and $dynamic\text{-}P_{20}$ use only 1.4% and 1.8% of the transactions used by TARMAQ, respectively).

These findings are further amplified when we look at the time required to extract the transactions needed to make the recommendation. As alluded to in the Introduction, TARMAQ generally takes little time to make a single recommendation, *provided* that the change history is readily available. However, extracting the history takes considerable time. For the systems considered in this study, TARMAQ was configured to use a modestly sized change history of 50,000 transactions, based on earlier experiments (Moonen et al., 2018). In contrast, the adaptive algorithms only need to extract enough of the change history to cover the number of *relevant* transactions for the given algorithm, which can be evaluated at extraction time and which needs proportionally less time and space than TARMAQ. For example, if the four most recent transactions are relevant, then $dynamic_{1,4}$ will only extract these four. Alternatively, if these four most recent *relevant* transactions are found in the last fifteen transactions, then $dynamic_{1,4}$ will only extract these fifteen transactions.

The extraction of TARMAQ's history of 50,000 transactions takes on average 203 minutes of CPU time and uses roughly 11MB of disk space per system after compression. This will not be an 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, these time and space requirements are linear in the number of projects to be considered, and to in-

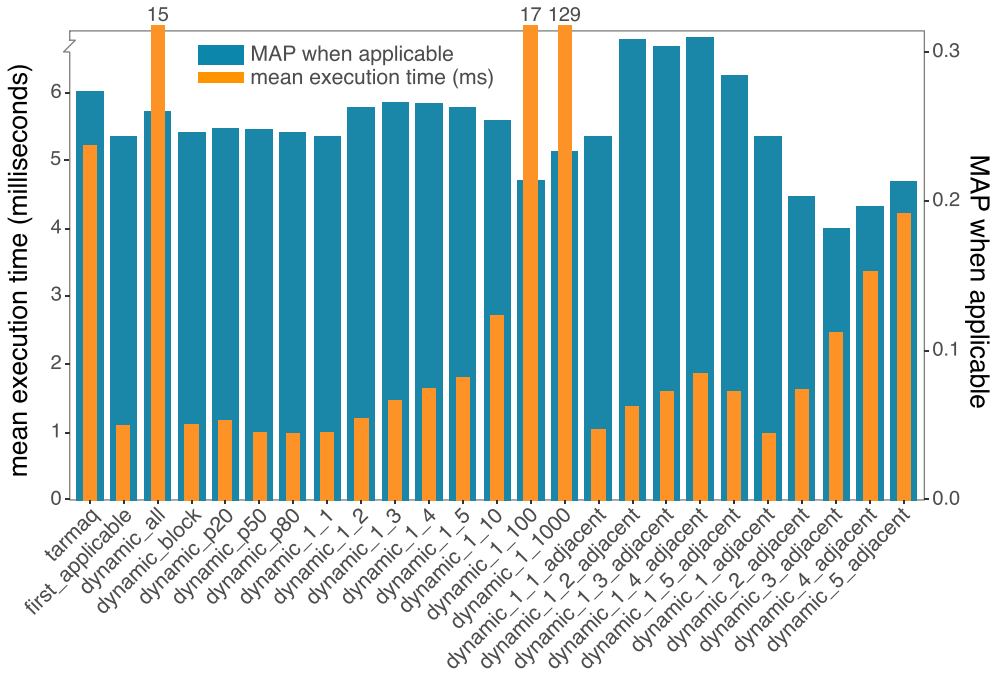

Fig. 9. Mean execution time of TARMAQ and the adaptive algorithms, together with the MAP value attained by each. Three outliers were truncated to keep a legible scale, their values are shown at the top of the plot.

vestigate the practical implications and opportunities of adaptive algorithms at a larger scale, we conduct a thought experiment that considers providing within-project change impact mining for an online service like GitHub.

In Sept. 2017, GitHub reported a total of 67 million repositories, of which 25.3 million are considered active (have seen activity in the preceding year). GitHub reported that the total number of repositories passed 100 million in November 2018, but since the number of active repositories has not been reported since 2017, we use the 25.3 million active repositories as a conservative estimate. Keeping a modest (50k) change history for just these active repositories would require 9765 years of CPU time for initial extraction and results in 272 terabytes of compressed data. While incremental extraction of new transactions will help keep the histories up-to-date, it will not solve the initial extraction effort, nor will it address space requirements. The same holds for on-demand extraction: if only 1% of the active projects would enable such recommendations, extraction still requires (on average) 97.65 years of CPU time. Parallelization will also help, but it takes a large number of cores to turn 97.65 years of CPU-time into a feasible time-span (e.g., it would take at least 5078 cores to reduce the extraction time for only 1% of active GitHub projects to a week, assuming optimal utilization and without accounting for increased communication overhead and bandwidth challenges). Moreover, during that week, the repositories will receive further updates that need to be accounted for.

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 min and require approximately 150KB of storage. 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 techniques.

8. Discussion

8.1. Trends

Some clear patterns emerge in the data. For example, TARMAQ-as-selector's impact improves when given a larger number of transactions to work with. In addition, age seems detrimental to accuracy. Finally, adjacency, while lowering applicability, brings value to the recommendation. These three 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 intricacy 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. Thus, this work raises the more introspective question "what about software enables only a few historical transactions to perform so well?"

8.2. Opportunities for hybrid algorithms

Table 4c shows that the *overall MAP* values of some of the adaptive algorithms come quite close to the MAP value attained by TARMAQ. This suggests the potential for hybrid techniques that combine TARMAQ with the other algorithms. One advantage that such hybrids bring is increased speed, especially when aiming to build efficient on-the-fly recommenders. This notion is supported by the timing data comparison shown in Fig. 9.

Taking a deeper look at Tables 3 and 5, it is interesting to note that in no case does TARMAQ always win, which is clear evidence that there is value in hybrid techniques that combine TARMAQ with the other algorithms to increase accuracy. For example, looking at

Table 4c, we can see that (numerically) there are four algorithms ($dynamic_{1,n}adjacent$, for $n = 2, 3, 4, 5$) with *MAP when applicable* values greater than that of TARMAQ. This means that a hybrid algorithm such as *"apply $dynamic_{1,2}adjacent$ if applicable, otherwise apply* TARMAQ" would match TARMAQ's applicability while exceeding its accuracy.

Moreover, comparing Table 4c's TARMAQ's *MAP when applicable* of 0.2744 to *dynamic_all*'s *MAP when applicable* of 0.2606 illustrates how greater selectivity regarding the transactions used to generate a recommendation brings increases accuracy. What is interesting here in terms of hybrid algorithms, is that the data in Table 3 includes *dynamic_all* winning 3775 times compared to TARMAQ's 3393, which shows that *dynamic_all* bests TARMAQ more often than not. One implication here is that on queries where TARMAQ does better, it does substantially better, and thus that there is value in being able to predict such queries.

Overall, the "B wins" column of Table 3 shows how often an adaptive algorithm performs better than TARMAQ at the individual query level. While in production, we lack an oracle to predict which algorithm to best use for a particular query, given the data from our experiments, we can compute the MAP value that a perfect prediction would achieve. This computation takes the mean of the highest AP value attained for each query by any of the algorithms (corresponding to a perfect selection of which algorithm to apply to each query). The MAP value computed for such a perfect prediction is 0.3798, which is a statistically significant improvement over the values produced by any of the individual algorithms. This (unattainable) MAP value does suggest the value of future research into hybrid algorithms.

8.3. MAP interpretation

The better algorithms studied have *MAP when applicable* values close to 0.33, while TARMAQ's *overall MAP* is 0.25. An AP value of 0.33 means that the first relevant artifact is found at rank 3, while 0.25 finds the first relevant artifact at rank 4. Because MAP represents a mean, for some queries, the rank of the first relevant artifact will be earlier in the ranked output, and for others, it will be later.

Consider an engineer using ATARI to check for missing artifacts. Optimistically the engineer is sufficiently familiar with the code to recognize a missing file or function based solely on its name. In this case, the engineer need only scan the top five to ten results to gain reasonable confidence that they have not missed any relevant artifacts.

Pessimistically the engineer is not sufficiently familiar with the code to sight identify missing artifacts. In this case, each artifact in the ranked list must be examined to determine its relevance. While this is clearly more costly then name-based identification, scanning several files or functions is not too arduous, and is clearly dwarfed by the cost of missing a modification and having to revisit the issue subsequently.

8.4. Threats to validity

Commits as a basis for evolutionary coupling: The evaluation in this paper is grounded in 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 (Herzig and Zeller, 2013; Herzig et al., 2016). Non-related artifacts may be present, and related artifacts may be missing from a transaction. We believe this threat is, at least, mitigated in the context of our study, as all but one of the systems (KM) uses 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 each 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 form 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 the relatedness of artifacts.

Equal weight for all commits: In our experiments, all transactions from the change history are given equal weight while mining change impact. A compelling alternative viewpoint is that more recent transactions are more relevant for current development and should, therefore, be given higher weight than older transactions. Similarly, 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 mitigates any differences between transactions by core developers and those of occasional contributors.

Multiple issue commits: Revision control system histories often include at least a small amount of noise. For example, commits related to a copyright update are noise in the context of uncovering evolutionary couplings. Another source of such noise is when a single commit addresses multiple issues. At present we do not attempt to identify such commits, although it might be possible to measure the cohesiveness of the committed artifacts by applying clustering algorithm or using a metric such as TF-IDF (Rajaraman and Ullman, 2011).

The inverse problem arises when two or more engineers work together to effect a change, and then each commits their change as a separate commit. At present ATARI is unable to learn cross-commit rules. While less of an issue with more modern version control systems, earlier research (Jaafar et al., 2014; Zimmermann et al., 2005; Kagdi et al., 2006) has worked on grouping commits from the change history that all address the same issue. Should this issue become a serious problem for ATARI, one of these approaches could be employed to combine such commits. Alternatively, this behavior might be seen as an asset if the separate commits reflect a code management style in which each engineer owns a particular part of the code, making cross-owner recommendations undesirable.

Variation in software systems: We conducted our experiments on two industrial systems and seventeen large open-source systems that, as illustrated in Table 2, were selected to vary considerably in both system- and change-history characteristics. 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.

9. Related work

The first algorithm for mining association rules was introduced in 1993, *AIS* (Agrawal-Imielinksi-Swami) (Agrawal et al., 1993). Since then, many improvements have been proposed, generally aimed at improving execution time or memory efficiency. These improvements can be classified into four major categories: (1) *Apriori* (Agrawal and Srikant, 1994), which uses an efficient precomputation of rule generation candidates, (2) *Eclat* (Zaki, 2000), which partitions the search space into smaller independent subspaces that can be analyzed efficiently, (3) *FPGrowth* (Han et al., 2004), 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) Arm (Rapid Association Rule Mining) (Das et al., 2001), 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* (Nath et al., 2013), 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 (Srikant et al., 1997; Hafez et al., 1999; Kubat et al., 2003). Targeted association rule mining ignores transactions unrelated to the query, which significantly reduces execution time. The adaptive algorithms studied in this paper aim at a commensurate reduction in the number of transactions considered, and consequently at a commensurate reduction in execution time.

More recently, Silva and Antunes presented an in-depth survey of constrained pattern mining (Silva and Antunes, 2016) 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. Fitting our work into their framework underscores the values of the approach and also suggests alternatives. The adaptive algorithms make use of several constraints. For example, adjacency is a sequencing constraint, while transaction relevance is a value constraint. Understanding the constraints and properties enables an algorithm designer to capture the semantics of the domain better, and thus helps to reduce the number of spurious results, as well as focus the algorithm on areas where it is more likely to gain information and return more interesting results.

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 (Gall et al., 1998; Beyer and Noack, 2005; Robbes et al., 2008), code-churn (Gall et al., 2003), and interactions with IDEs (Zanjani et al., 2014).

For software change impact analysis, common algorithms such as Rose (Zimmermann et al., 2005), apply targeted association rule mining to derive change recommendations given a user-specified query. Rose, similar to other software change recommendation approaches subsequently introduced (Wang et al., 2009; Bavota et al., 2013), is based on the Apriori algorithm. Ying et al. (Ying et al., 2004) describe a technique that mines frequent patterns in a systems' change history to recommend potentially relevant code to a developer performing a software maintenance task.

Several projects have considered aspects of the mining problem that, to varying degrees, compliment the investigation of adaptive algorithms. For example, recent research highlighted how the configuration parameters of data mining algorithms have a significant impact on the quality of the results (Maimon and Rokach, 2010). In the context of association rule mining, several authors have highlighted the need for thoughtfully studying how parameter settings affect the quality of the rules generated (Zheng et al., 2001; Lin et al., 2002; Jiang and Gruenwald, 2006). For example, Moonen et al. recently investigated how the quality of software change recommendation varied depending on association rule mining parameters such as the transaction filtering threshold, history length, and history age (Moonen et al., 2016; 2018). 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.

A complimentary aspect concerns the grouping of a project's transactions (Jaafar et al., 2014; Zimmermann et al., 2005; Kagdi et al., 2006). The reason for doing so is that a developer might accidentally commit an incomplete transaction and commit the remaining files related to the same change in a subsequent transaction. As a consequence, the artifacts related to a single logical change might be spread across several transactions in the change history. This type of grouping may not be needed when using a version control system where changes are first stored locally, and then distributed at a later stage, as the option to revise and combine local transactions reduces the risk of committing incomplete transactions.

Recently Islam et al. (2018) presented an algorithm to capture indirect coupling. For example, in order to recommend a, prior algorithms require that a co-changes with at least one element of the query in the history. Islam et al. consider the possibility that artifacts that did not directly co-change in the past might still be related. To uncover such artifacts, they consider *transitive association rules*. For example, if a and b co-change in the history, and so do b and c, then it becomes possible to recommend a given a query that involves c.

Finally, the software repository mining literature (Graves et al., 2000; Zimmermann et al., 2005; Hassan, 2008) frequently alludes to the notion that learning from too short, or overly long history harms the outcome, either because not enough knowledge can be uncovered, or because outdated information introduces noise. Moonen et al. (2018) 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.

10. 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 a vanishingly small amount of the history. Doing so enables us to take a finer-grained look at understanding the value that 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. Given how little of the history the adaptive techniques make use of, their success suggests the value in considering software-specific association rule mining algorithms.

Contributions: This paper makes the following five contributions. (1) It introduces the notion of adaptive targeted association rule mining. (2) It proposes several families of adaptive algorithms

for change impact mining. (3) It implements these algorithms in the prototype tool ATARI. (4) It compares the new algorithms to each other and to the state-of-the-art tool, TARMAQ (Rolfsnes et al., 2016). (5) Finally, it studies the impact of applying TARMAQ as a filter on the transactions selected by each adaptive algorithm.

Future work: Looking forward, we see several interesting directions for future work. To begin with, it would be interesting to attempt to provide a natural language explanation supporting each recommendation. Given that our recommendations are rule-based, it should be feasible to synthesize an explanation from the rules that were applied.

In addition, as discussed in Section 8, the existence of low applicability and high *MAP when applicable* algorithms suggests the potential value in creating *hybrid algorithms*. For example, our results suggest there is potential for using machine learning to create an *ensemble* of algorithms. As a preliminary experiment, a hand-crafted ensemble applied $dynamic_{1,2}adjacent$ if it is applicable and TARMAQ otherwise. The ensemble takes advantage of $dynamic_{1,2}adjacent$'s high MAP while maintaining TARMAQ's high applicability.

Another area of future work concerns rule-aggregation, which combines the evidence of two or more rules (Rolfsnes et al., 2018). Aggregation likely increases recommendation relevance: for example, consider three rules, one recommending *A* with confidence 0.8 and two recommending *B* with confidence 0.7 and 0.6, respectively. Traditional approaches use the highest ranked rule and thus prioritize *A* over *B*. Rule aggregation combines the evidence for *B* and thus leads to recommending *B* over *A*. In theory this is expected to bring less value, given that the adaptive algorithms make use of fewer transactions; however, it might still prove useful. Moreover, it would be of interest to take a closer look at the relative impact of *transaction age* (Moonen et al., 2018) and adjacency on recommendation accuracy.

Last but not least, the viability of making predictions based on very few transactions 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 is warranted.

Acknowledgments

This work is supported by the Research Council of Norway (FRN) through the EvolveIT project (#221751/F20) and the Certus SFI (#203461/030). Dr. Binkley is supported by NSF grants IIA-1360707 1626262, and a J. William Fulbright award. 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

Agrawal, R., Imielinski, T., Swami, A., 1993. Mining association rules between sets of items in large databases. In: ACM SIGMOD International Conference on Management of Data. ACM, pp. 207–216. doi:10.1145/170035.170072.

Agrawal, R., Srikant, R., 1994. Fast algorithms for mining association rules. In: International Conference on Very Large Data Bases (VLDB), pp. 487–499.

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

Baeza-Yates, R., Ribeiro-Neto, B., 1999. Modern Information Retrieval. Addison-Wesley.

Bavota, G., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D., De Lucia, A., 2013. An empirical study on the developers' perception of software coupling. In: 2013 35th International Conference on Software Engineering (ICSE). IEEE, pp. 692–701. doi:10.1109/icse.2013.6606615.

Beyer, D., Noack, A., 2005. Clustering Software artifacts based on frequent common changes. In: International Workshop on Program Comprehension (IWPC). IEEE, pp. 259–268. doi:10.1109/wpc.2005.12.

Bohner, S., Arnold, R., 1996. Software Change Impact Analysis. IEEE, CA, USA.

Canfora, G., Cerulo, L., 2005. Impact analysis by mining software and change request repositories. In: International Software Metrics Symposium (METRICS). IEEE, pp. 29–37. doi:10.1109/metrics.2005.28.

Collard, M.L., Decker, M.J., Maletic, J.I., 2013. srcML: an infrastructure for the exploration, analysis, and manipulation of source code: a tool demonstration. In: IEEE International Conference on Software Maintenance (ICSM). IEEE, pp. 516–519. doi:10.1109/icsm.2013.85.

Das, A., Ng, W.-K., Woon, Y.-K., 2001. Rapid association rule mining. In: Proceedings of the Tenth International Conference on Information and Knowledge Management - CIKM'01. ACM Press, New York, New York, USA, p. 474. doi:10.1145/502585.502665.

Eick, S., Graves, T.L., Karr, A., Marron, J., Mockus, A., 2001. Does code decay? assessing the evidence from change management data. IEEE Trans. Softw. Eng. 27 (1), 1–12. doi:10.1109/32.895984.

Gall, H., Hajek, K., Jazayeri, M., 1998. Detection of logical coupling based on product release history. In: IEEE International Conference on Software Maintenance (ICSM). IEEE, pp. 190–198. doi:10.1109/icsm.1998.738508.

Gall, H., Jazayeri, M., Krajewski, J., 2003. CVS release history data for detecting logical couplings. In: International Workshop on Principles of Software Evolution (IWPSE). IEEE, pp. 13–23. doi:10.1109/iwpse.2003.1231205.

Gethers, M., Kagdi, H., Dit, B., Poshyvanyk, D., 2011. An adaptive approach to impact analysis from change requests to source code. In: IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, pp. 540–543. doi:10.1109/ase.2011.6100120.

Graves, T.L., Karr, A., Marron, J., Siy, H.P., 2000. Predicting fault incidence using software change history. IEEE Trans. Softw. Eng. 26 (7), 653–661. doi:10.1109/32.859533.

Hafez, A., Deogun, J., Raghavan, V.V., 1999. The item-set tree: a data structure for data mining. In: Data Warehousing and Knowledge Discovery. In: LNCS, 1676. Springer, pp. 183–192. doi:10.1007/3-540-48298-9.

Han, J., Pei, J., Yin, Y., Mao, R., 2004. Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min Knowl Discov 8 (1), 53–87. doi:10.1023/b:dami.0000005258.31418.83.

Hassan, A.E., 2008. The road ahead for Mining Software Repositories. In: Frontiers of Software Maintenance. IEEE, pp. 48–57. doi:10.1109/fosm.2008.4659248.

Hassan, A.E., Holt, R., 2004. Predicting change propagation in software systems. In: IEEE International Conference on Software Maintenance (ICSM). IEEE, pp. 284–293. doi:10.1109/icsm.2004.1357812.

Herzig, K., Just, S., Zeller, A., 2016. The impact of tangled code changes on defect prediction models. Empir. Softw. Eng. 21 (2), 303–336. doi:10.1007/s10664-015-9376-6.

Herzig, K., Zeller, A., 2013. The impact of tangled code changes. In: Working Conference on Mining Software Repositories (MSR). IEEE, pp. 121–130. doi:10.1109/msr.2013.6624018.

Islam, M.A., Islam, M.M., Mondal, M., Roy, B., Roy, C.K., Schneider, K., 2018. Detecting evolutionary coupling using transitive association rules. In: Proceedings of 2018 IEEE Workshop on Source Code Analysis and Manipulation (SCAM'18). Madrid, Spain.

Jaafar, F., Guéhéneuc, Y.-G., Hamel, S., Antoniol, G., 2014. Detecting asynchrony and dephase change patterns by mining software repositories. J. Softw. 26 (1), 77–106. doi:10.1002/smr.1635.

Jashki, M.-A., Zafarani, R., Bagheri, E., 2008. Towards a more efficient static software change impact analysis method. In: ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE). ACM, pp. 84–90. doi:10.1145/1512475.1512493.

Jiang, N., Gruenwald, L., 2006. Research issues in data stream association rule mining. ACM SIGMOD Record 35 (1), 14–19. doi:10.1145/1121995.1121998.

Kagdi, H., Gethers, M., Poshyvanyk, D., 2013. Integrating conceptual and logical couplings for change impact analysis in software. Empir. Softw. Eng. 18 (5), 933–969. doi:10.1007/s10664-012-9233-9.

Kagdi, H., Yusuf, S., Maletic, J.I., 2006. Mining sequences of changed-files from version histories. In: International Workshop on Mining Software Repositories (MSR). ACM, pp. 47–53. doi:10.1145/1137983.1137996.

Kubat, M., Hafez, A., Raghavan, V.V., Lekkala, J., 2003. Itemset trees for targeted association querying. IEEE Trans. Knowl. Data Eng. 15 (6), 1522–1534. doi:10.1109/tkde.2003.1245290.

Law, J., Rothermel, G., 2003. Whole Program Path-Based Dynamic Impact Analysis. In: International Conference on Software Engineering (ICSE). IEEE, pp. 308–318.

Lin, W., Alvarez, S.A., Ruiz, C., 2002. Efficient adaptive-support association rule mining for recommender systems. Data Min. Knowl. Discov. 6 (1), 83–105. doi:10.1023/a:1013284820704.

Maimon, O., Rokach, L., 2010. Data Mining and Knowledge Discovery Handbook. Springer doi:10.1007/978-0-387-09823-4.

Moonen, L., Di Alesio, S., Binkley, D., Rolfsnes, T., 2016. Practical guidelines for change recommendation using association rule mining. In: International Conference on Automated Software Engineering (ASE). ACM, pp. 732–743. doi:10.1145/2970276.2970327.

Moonen, L., Rolfsnes, T., Binkley, D., Di Alesio, S., 2018. What are the effects of history length and age on mining software change impact? Empir. Softw. Eng. (EMSE) (https://doi.org/10.1007/s10664-017-9588-z) 1–36. doi:10.1007/s10664-017-9588-z.

Nath, B., Bhattacharyya, D.K., Ghosh, A., 2013. Incremental association rule mining: a survey. Wiley Interdiscip. Rev. 3 (3), 157–169. doi:10.1002/widm.1086.

Podgurski, A., Clarke, L., 1990. A formal model of program dependences and its implications for software testing, debugging, and maintenance. IEEE Trans. Softw. Eng. 16 (9), 965–979. doi:10.1109/32.58784.

Pugh, S., Binkley, D., Moonen, L., 2018. The Case for Adaptive Change Recommendation, 18th IEEE International Working Conference on Source Code Analysis and

Manipulation, SCAM 2018, Madrid. September 23–24, IEEE Computer Society, Spain.
Rajaraman, A., Ullman, J.D., 2011. Mining of Massive Datasets. Cambridge University Press, New York, NY, USA.
Ren, X., Shah, F., Tip, F., Ryder, B.G., Chesley, O., 2004. Chianti: a tool for change impact analysis of java programs. In: ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), pp. 432–448. doi:10.1145/1035292.1029012.
Robbes, R., Pollet, D., Lanza, M., 2008. Logical Coupling Based on Fine-Grained Change Information. In: Working Conference on Reverse Engineering (WCRE). IEEE, pp. 42–46. doi:10.1109/wcre.2008.47.
Rolfsnes, T., Di Alesio, S., Behjati, R., Moonen, L., Binkley, D.W., 2016. Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis. In: International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE, pp. 201–212. doi:10.1109/saner.2016.101.
Rolfsnes, T., Moonen, L., Alesio, S.D., Behjati, R., Binkley, D., 2018. Aggregating Association Rules to Improve Change Recommendation. Empir. Softw. Eng. (EMSE) 23 (2), 987–1035. doi:10.1007/s10664-017-9560-y.
Rolfsnes, T., Moonen, L., Di Alesio, S., Behjati, R., Binkley, D., 2017. Improving change recommendation using aggregated association rules. J. Empir. Softw. Eng. (EMSE).
Silva, A., Antunes, C., 2016. Constrained pattern mining in the new era. Knowl. Inf. Syst. 47 (3).
Srikant, R., Vu, Q., Agrawal, R., 1997. Mining association rules with item constraints. In: International Conference on Knowledge Discovery and Data Mining (KDD). AASI, pp. 67–73.
Tan, P.-N., Kumar, V., Srivastava, J., 2002. Selecting the right interestingness measure for association patterns. In: International Conference on Knowledge Discovery and Data Mining (KDD). ACM, p. 32. doi:10.1145/775052.775053.
Wang, X., Wang, H., Liu, C., 2009. Predicting co-changed software entities in the context of software evolution. In: Proceedings - 2009 International Conference on Information Engineering and Computer Science, ICIECS 2009 doi:10.1109/iciecs.2009.5364521.
Yazdanshenas, A.R., Moonen, L., 2011. Crossing the boundaries while analyzing heterogeneous component-based software systems. In: IEEE International Conference on Software Maintenance (ICSM). IEEE, pp. 193–202. doi:10.1109/icsm.2011.6080786.
Ying, A.T.T., Murphy, G., Ng, R.T., Chu-Carroll, M., 2004. Predicting source code changes by mining change history. IEEE Trans. Softw. Eng. 30 (9), 574–586. doi:10.1109/tse.2004.52.
Zaki, M.J., 2000. Scalable algorithms for association mining. IEEE Trans Knowl Data Eng 12 (3), 372–390. doi:10.1109/69.846291.
Zanjani, M.B., Swartzendruber, G., Kagdi, H., 2014. Impact analysis of change requests on source code based on interaction and commit histories. In: International Working Conference on Mining Software Repositories (MSR), pp. 162–171. doi:10.1145/2597073.2597096.
Zheng, Z., Kohavi, R., Mason, L., 2001. Real world performance of association rule algorithms. In: SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). ACM, pp. 401–406. doi:10.1145/502512.502572.
Zimmermann, T., Zeller, A., Weissgerber, P., Diehl, S., 2005. Mining version histories to guide software changes. IEEE Trans. Softw. Eng. 31 (6), 429–445. doi:10.1109/tse.2005.72.