Debugging Crashes using *Continuous* Contrast Set Mining

Rebecca Qian*
Facebook, Inc.
U.S.A.
rebeccaqian@fb.com

Yang Yu
Purdue University
U.S.A.
yu577@purdue.edu

Wonhee Park
Facebook, Inc.
U.S.A.
wonheepark@fb.com

Vijayaraghavan Murali
Facebook, Inc.
U.S.A.
vijaymurali@fb.com

Stephen Fink
Facebook, Inc.
U.S.A.
stephenfink@fb.com

Satish Chandra
Facebook, Inc.
U.S.A.
schandra@acm.org

ABSTRACT

Facebook operates a family of services used by over two billion people daily on a huge variety of mobile devices. Many devices are configured to upload crash reports should the app crash for any reason. Engineers monitor and triage millions of crash reports logged each day to check for bugs, regressions, and any other quality problems. Debugging groups of crashes is a manually intensive process that requires deep domain expertise and close inspection of traces and code, often under time constraints.

We use contrast set mining, a form of discriminative pattern mining, to learn what distinguishes one group of crashes from another. Prior works focus on discretization to apply contrast mining to continuous data. We propose the first direct application of contrast learning to continuous data, without the need for discretization. We also define a weighted anomaly score that unifies continuous and categorical contrast sets while mitigating bias, as well as uncertainty measures that communicate confidence to developers. We demonstrate the value of our novel statistical improvements by applying it on a challenging dataset from Facebook production logs, where we achieve 40x speedup over baseline approaches using discretization.

CCS CONCEPTS

• **Software and its engineering → Software reliability**.

KEYWORDS

crash analysis, descriptive rules, rule learning, contrast set mining, emerging patterns, subgroup discovery, multiple hypothesis testing

ACM Reference Format:
Rebecca Qian, Yang Yu, Wonhee Park, Vijayaraghavan Murali, Stephen Fink, and Satish Chandra. 2020. Debugging Crashes using *Continuous* Contrast Set Mining. In *Software Engineering in Practice (ICSE-SEIP '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3377813.3381369

1 INTRODUCTION

In commercial software development, despite significant investment in software quality processes including static and dynamic analyses, code reviews and testing, defects still slip through and cause crashes in the field. Fixing these crashes remains a manually intensive process, demanding deep domain expertise and detailed analysis of traces and code.

Large software organizations that develop mobile apps often deploy automated crash triage systems, which capture error logs when a mobile client crashes. These logs contain hundreds of key-value pairs with metadata about the app's execution environment, such as the mobile OS version or app build, and possibly a trace of where a crash occurred. Once captured, an automated system usually groups crash logs into categories (e.g., by a hash on a descriptive value, or through more sophisticated clustering) and then assigns each category to on-call developers. If categorization were perfect, each crash in a category would arise from the same root cause.

Often, developers trying to resolve a group of crashes want to know what distinguishes a particular group of crashes. For instance, developers ask: "does this group of crashes occur disproportionately in build version X?", or "does this group of crashes occur disproportionately for users from country Y?". One simple way to describe a group of crashes is to use standard statistical tests regarding the distribution of features among members of the group. For example, one could test if `country:Y` appears statistically more frequently in one group of crashes than in the whole population.

One limitation of standard statistical tests is that they may not reveal patterns involving interactions among multiple features. Our crash data includes many dimensions of features, with a mix of categorical, discrete, and continuous values. Additionally, we need to generate *interpretable* insights that a human can comprehend, which rules out standard dimensionality reduction techniques (*e.g.* principal component analysis) which tend to compute complex, unintuitive factors.

Recently, Castelluccio *et al.* [4] proposed using *contrast set mining* to extract insights from multi-dimensional data about crashes. Contrast set mining (CSM) [2] is a form of discriminative pattern mining that attempts to discover significant patterns that occur with disproportionate frequencies in different groups. It explores the space of feature sets, i.e., sets of conjunctive feature-value pairs, looking for deviations from expected distributions. For example, the feature set $\{\mathsf{build\_version} : \mathsf{X}, \mathsf{country} : \mathsf{Y}\}$ is interpreted as the value of the build version feature being X and country being Y. CSM models the expected distributions of these sets from the general population of data points, i.e., independent of any particular group. Then, given a particular group of points, if the distribution of a feature set in that group differs significantly from its expected value, it is labeled as a *contrast set*. The notion of differing significantly is defined by an explicit statistical test, and denotes a degree to which the contrast set is anomalous. Castelluccio *et al.* 's application of CSM produced relevant hints to developers regarding groups of user-reported bugs, helping them fix bugs faster. It also helped uncover systematic breakages by detecting anomalous attribute-value pairs.

Thus far, most applications of CSM consider only categorical data. When applying CSM to discrete and continuous variables, researchers typically *discretize* data by sorting values into buckets, resulting in a limited number of possible states. For example, $\{\text{process\_uptime} : (0, 2000)\}$ represents a discretized feature that denotes instances where a process ran for less than 2000 milliseconds. Discretization has notable drawbacks; it does not scale well to large datasets with thousands of features as it leads to an explosion in the number of feature-value pairs. Moreover, if the feature ranges are strongly skewed, the discretized bins do not capture the distribution well. Discretized ranges are also often unintuitive, causing resulting contrast sets to be difficult to interpret.

To address these drawbacks, we propose several improvements to CSM to extend it to continuous features and other mixed data types, such as event sequences, *without discretization.* We demonstrate its effectiveness by applying it to a class of hard bugs: app deaths from iOS out-of-memory crashes (OOMs). These OOMs are hard to resolve since they do not provide logs with stack traces. Instead, they are annotated with *user navigation logs*, i.e., sequences of events that a user navigated, ordered chronologically, before the crash occurred. As we will describe later, we compute a continuous vector-space encoding of these navigation logs as a technique to enable tractable analysis of this high dimensional data, while still enabling accurate localized information about a potential root cause.

Towards these goals, our key contributions are the following:

- We propose Continuous Contrast Set Mining (CCSM), the first direct application of contrast mining to continuous data without discretization.
- We evaluate its effectiveness on a dataset containing $60k$ iOS OOM crashes: using CCSM, on average we generated 1120 contrast sets in 458 seconds. This is a 40x speedup over a naive discretization approach. We also evaluate the usefulness of CCSM towards debugging software issues in an industrial setting.
- We provide a formal definition of contrast set quality, allowing us to rank and compare categorical and continuous contrast sets while mitigating bias.
- We propose uncertainty measures based on confidence intervals evaluating effect size and difference in means to provide signals to developers on actionability.

The rest of the paper is organized as follows. Section 2 gives an overview of the previously proposed algorithm for CSM and its limitations when working with continuous features. Section 3 describes our proposed algorithm, CCSM, that handles continuous features and a definition of anomaly score that unifies categorical and continuous contrast sets. Section 4 discusses how contrast sets from CCSM can aid software debugging in practice in the industry. Section 5 presents experimental results evaluating our algorithm, including preliminary experience with it. Section **??** describes threats to the validity of our study. Finally, Sections 6 and 7 discuss related works and future directions, respectively.

2 OVERVIEW

In this section, we describe the CSM problem and summarize the existing algorithm for CSM on categorical variables, namely STUCCO [2]. We then discuss what kind of continuous features arise in crashes and limitations of using discretization to apply STUCCO to the continuous domain.

2.1 STUCCO Contrast Set Mining Algorithm

The objective of CSM is to find statistically meaningful differences between groups. In CSM, we start with a categorical dataset partitioned into mutually exclusive groups. A candidate contrast set is a conjunction of attribute-value pairs, where an attribute is a database field and the value is one of a range of values that field can take on, *e.g.* `country=IN`.

For a contrast set X and group G, the support $S(X, G)$ is the *percentage of vectors in group G for which the contrast set X is true.* We want to find "interesting" contrast sets whose support differs *meaningfully* across groups. For differences in support to be meaningful, the contrast set must be both *significant* and *large.* More formally, contrast sets must satisfy two conditions,

$$\exists ij \text{ s.t. } P(X|G_i) \neq P(X|G_j) \tag{1}$$

and

$$max_{ij}|S(X, G_i) - S(X, G_j)| \geq \delta \tag{2}$$

where $P(X|G_i)$ is the likelihood of observing set X for group G_i and δ is the user defined *minimum support difference.* Equation (1) tests a contrast set for statistical significance, and Equation (2) checks for largeness, i.e., that the support of the contrast set differs by a certain threshold for at least two groups.

For the baseline, we implemented the STUCCO algorithm, which casts contrast set mining as a tree search problem [2]. Starting with an empty root node, we begin by enumerating all attribute-value pairs in the dataset. Figure 1 shows the initial candidates for a toy dataset with two columns (*country*, *os_version*) that each can take on two values.

For each candidate node, we scan the dataset and count support for each group. We then examine whether the node is significant and large. In STUCCO, a contrast set is statistically significant if it passes a two-way Chi Squared test. The *null hypothesis* is that the support of a contrast set is equal across all groups, i.e., independent of group membership. The Chi Squared test evaluates this hypothesis by analyzing expected and observed frequencies, taking into account factors such as the number of observations, group sizes and variance.

Nodes that fail either condition (1) or (2) are pruned. After testing all candidates in a given level, we generate children from the surviving nodes by forming conjunctions of attributes. We use

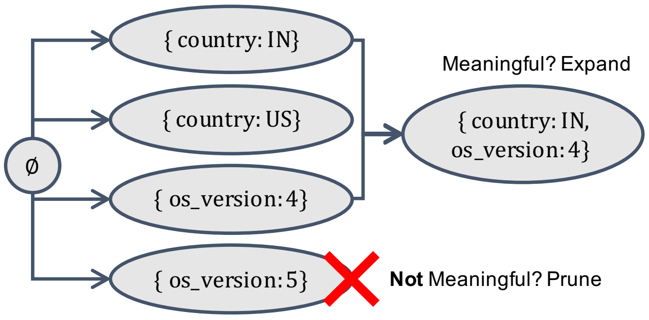


Figure 1: Sample generation of initial candidate sets

a canonical ordering of attributes to avoid redundant combinations. For example, assuming no nodes are pruned, the children of contrast set {country : IN} are {country : IN, os_version : 4.0} and {country : IN, os_version : 5.0}.

The generated child nodes become the new candidate set. In addition to significant and large conditions, we implement a variety of pruning techniques as described by Castelluccio *et al.* [4]. We repeat the above process of testing and generating contrast sets, until no new child nodes can be formed. As in [4], we reduce the likelihood of type 1 errors (false positives) in statistical testing by applying the Bonferroni correction, which lowers critical values as the tree depth increases.

Contrast set mining has several advantages when compared to other techniques used to mine feature sets, such as Decision Trees. Compared to CSM, limitations of Decision Tree algorithms include the lack of statistical significance testing, which does not provide guarantees on split quality. Additionally, as with other greedy algorithms, the order of decisions impacts the results.

Next, we describe the setting in which we wish to apply CSM, and the limitations of the standard STUCCO algorithm in this setting.

2.2 Continuous Features of Crash Reports

With billions of active mobile users, Facebook must monitor and maintain the health of mobile apps at huge scale. When a crash occurs, a snapshot of the mobile client and app level information is logged into a crash report, which is then uploaded to a server. A crash report often includes the stack trace associated with the crash, which is one of the most important signals for a developer debugging the crash.

For certain classes of crashes, however, the stack trace is unavailable or difficult to obtain. For instance, when an out-of-memory error (OOM) occurs in unmanaged code, the OS kills the app and does not have memory to snapshot a stack trace. In other crashes from native code, the stack trace may not contain debugging symbols and conveys little interpretable information. For these classes of crashes, termed "hard bugs", developers can only rely on other features and metadata when debugging.

Table 1 shows a subset of the device metadata features that are logged in crash reports. One feature that developers find particularly useful for dealing with hard bugs are *navigation logs*. A navigation log is a sequence of app surfaces that a user interacted with prior to experiencing the crash. Figure 2 shows an example of a navigation log where the user transitioned from the Feed surface to Friends. Bi-grams extracted from a navigation log show the source and destination surface of a single navigation event. These bi-grams help localize crash insights to certain parts of the whole sequence and help reasoning about individual navigation events.

Table 1: Sample metadata collected by crash error logs

Attribute	Explanation	Type
Build ID	Build number of the crashing app	Categorical
OS Version	Version of the mobile operating system	Categorical
Fd count	Number of open file descriptors	Discrete
Country	Country associated with the mobile device	Categorical
Process uptime	Time since app process started	Continuous
Nav logs	Event navigation sequences before crash	Sequential
Bi-grams in nav logs	TF-IDF vectorization of bi-grams in nav logs	Continuous

Figure 2: Contrived example of a navigation log and its continuous bi-gram features

$\boxed{\text{Feed} \longrightarrow \text{Photos} \longrightarrow \text{Fundraiser} \longrightarrow \text{Feed} \longrightarrow \text{Photos} \longrightarrow \text{Friends}}$

Bi-gram	TF-IDF weight
Feed ⟶ Photos	$2 * 0.01$
Photos ⟶ Fundraiser	$1 * 9.85$
Fundraiser ⟶ Feed	$1 * 7.42$
Photos ⟶ Friends	$1 * 1.25$
Video ⟶ Feed	$0 * 0.05$
...	0

For instance, certain navigation events (bi-grams) tend to occur more commonly than others – say, navigating to or from Feed is more common than Fundraiser. Given a navigation log, this information can be quantified by using the TF-IDF weight of the bi-grams in the log. TF-IDF [9] is a well-known method in information retrieval to filter the more important features of textual documents from noise. The TF-IDF weight of a bi-gram denotes how often it appears in the entire corpus of navigation events as opposed to a particular log. Figure 2 illustrates an example. A high weight for a bi-gram indicates that it is more important to this navigation log, i.e., less common in the entire corpus. Thus, each bi-gram in the corpus can be considered a feature that can take on any positive real value for each navigation log.

While many features in Table 1, such as country, are categorical (i.e., their values come from a finite set), TF-IDF encoded bi-gram features are continuous. Categorical features are quite amenable to CSM, whereas continuous features pose several challenges to traditional CSM.

2.2.1 The Continuous Problem. The original STUCCO algorithm [2] for CSM generates contrast sets based on *categorical* features, such as user locale or CPU model, and Castelluccio *et al.* [4] applied it on such features. In many real world settings, however, crash reports include both categorical variables such as country, and numerical variables (discrete or continuous) such as file descriptor counts and memory usage. Most applications of CSM to continuous features rely on entropy-based discretization methods, i.e., splitting the continuous domain into discrete intervals and treating them as categorical values. Following the seminal STUCCO algorithm, Bay proposed an initial data discretization method for CSM [1]. Simeon and Hilderman proposed a slightly modified equal width binning interval to discretize continuous variables [10].

In practice, however, discretization of continuous features has several drawbacks. First, it greatly increases the number of candidate contrast sets. Each discretized bin results in a new candidate contrast set, and computation can become prohibitively expensive with a large number of continuous features. Section 5 presents empirical results which quantify this computational cost.

Second, discretizing continuous data may yield results that are difficult to interpret. Figure 3 shows the histogram of TF-IDF scores of navigation event sequences. Let's consider the case that we use equal-width bins of width 0.5, and find an arbitrary set of bins, say (0, 0.5), (2, 2.5), and (5, 5.5), are statistically significant, but not other intervals. Developers may not find these results actionable, as the results may reflect the choice of cut-off points more than underlying patterns in the data.

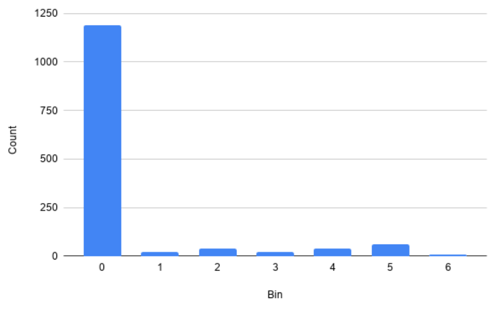


Figure 3: Frequency Distribution of TF-IDF encodings of event sequences

Finally, any form of discretization leads to information loss, especially at the tails. In figure 3, the strong skew in distribution with a large proportion of zero values may drive discretization, with the second smaller hump at (4.5, 5.5) going unnoticed. In this case, most discretized contrast sets cannot represent the magnitude of mean differences between two groups, even if mean difference provides important debugging information to developers. In our context, if repeated navigation events that are rare overall lead to out-of-memory crashes, mean difference in number of navigation events would convey important debugging context without information loss.

These drawbacks of discretization provide motivation for developing a continuous version of contrast set mining.

3 CONTINUOUS CONTRAST SET MINING

Addressing the limitations just described, we propose the CCSM algorithm, which applies CSM directly to continuous data. Additionally, we define separate and unified anomaly scores for continuous and categorical contrast sets. Recognizing that real world datasets are frequently mixed, our *unified anomaly score* is the first ranking algorithm that produces a normalized comparison of the two anomaly definitions. Finally, we describe confidence intervals on contrast sets, and how we translate them into interpretable findings.

3.1 Base CCSM Algorithm

The CCSM algorithm adopts the same structure as STUCCO described earlier in Section 2, with modifications to reason about sets of continuous attributes. As previously discussed, STUCCO requires discretization to handle continuous or discrete features with numerical ranges. The CCSM algorithm instead reasons directly about continuous contrast sets without introducing discretized bins.

For CCSM, the input consists of a set of k-dimensional numerical vectors, where k is the number of continuous variables, partitioned into mutually exclusive groups (SIGs). A contrast set is either a single continuous variable or a set of continuous variables. We start by considering single continuous variables.

As in the original STUCCO algorithm, we consider a contrast set a deviation if it is both *significant* and *large*. We develop counterparts for these two conditions in the continuous domain.

We define a contrast set to be *significant* or statistically significant using *one-way ANOVA F-test*. One-way ANOVA F-test is used to test whether there is a significant difference in the mean of a continuous variable across groups. Applied to our context, one-way ANOVA F-test tests the null hypothesis that the mean of the contrast set is the same across all groups and rejects the null when there are at least two groups with a significant difference in mean. This is a natural counterpart to the *Pearson's chi-squared test* used to identify significant contrast sets in the original STUCCO algorithm, which tests the null hypothesis that the percentage of the contrast set is the same across all groups. As with the STUCCO algorithm, we apply a set of pruning heuristics and apply the Bonferroni correction to reduce the likelihood of type 1 errors in tree-based search and testing.

We define a contrast set to be *large* or practically significant if there exist two groups such that the difference of the average values of the contrast set in these two groups is greater than some user-defined threshold δ. This definition also mirrors that in the original STUCCO algorithm, where a contrast set is defined to be large if the percentage difference of a contrast set in two groups is larger than some threshold.

Given these definitions of *significant* and *large*, we apply the STUCCO tree search algorithm to efficiently search for conjunctions of contrast sets that distinguish a particular group of vectors (a SIG) from the rest of the population. Algorithm 1 shows pseudo-code for the CCSM algorithm, with the base algorithm starting on line 9. Algorithm 1 additionally includes some details specific to mining navigation logs – subsequent sections discuss these details.

3.2 Ranking Contrast Sets

We use Cohen's d as the anomaly score for continuous contrast sets. Cohen's d is a measure of effect size, or more specifically, a measure of the difference between two group means. The formula is given by

$$d = \frac{\bar{x}_1 - \bar{x}_2}{s}$$

Algorithm 1 Continuous Contrast Set Mining on Navigation Sequences

```
1:  procedure CCSM ALGORITHM
2:      $Q \leftarrow$ initial candidate set of n-grams in $S$
3:      Result set $R \leftarrow \emptyset$
4:      while $Q$ is not empty do
5:  preprocess:
6:          for each $q$ in $Q$ do
7:              $IDF(q) = ln(\frac{docCount - f(q) + 0.5}{f(q) + 0.5})$
8:              count $f(q,d)$ for each FAD $d \in D$
9:  CCSM:
10:         for each $q$ in $Q$ do
11:             is_significant $\leftarrow ANOVA(q, X)$
12:             means_difference $\leftarrow \max_{i,j} |mean(q|G_i) - mean(q|G_j)|$
13:             if $prune(q)$ is True then
14:                 continue
15:             if $is\_significant \land means\_difference > \delta$ then
16:                 append $q$ to $R$
17:         $Q \leftarrow$ gen_candidates($R$)
18:     return $R$
```

Table 2: Qualitative evaluations of Effect Size

Magnitude	Effect Size (d)
very small	0.01
small	0.2
medium	0.5
large	0.8
very large	1.2
huge	2

where $\bar{x}_1$ and $\bar{x}_2$ are the two sample means, and s is the pooled standard deviation, defined as

$$s = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}}$$

where s_1 and s_2 are the two sample standard deviations. The anomaly score can be seen as the standardized mean difference on contrast set X between group A and all the other groups. If we see $d = 1$, we know that the two means differ by one standard deviation; $d = 0.5$ tells us that the two means differ by half a standard deviation, and so on. An advantage of Cohen's d over *two-sample t-statistic*, another measure of standardized difference in means, is that Cohen's d is not as heavily affected by sample size, because it measures the effect size relative to the standard deviations of groups rather than relative to the standard error of the mean difference, which shrinks as the sample size grows.

Table 2 shows how we convert measures of effect size into descriptive interpretations that can be displayed in crash analysis UI. The higher the anomaly score is (which means X has larger values inside A than outside A), the more prominent the contrast set X is to group A.

For every group, we score each CCSM contrast set by its anomaly score. We then submit the top deviations for each group, ranked by their anomaly scores.

3.2.1 *Unifying Continuous and Categorical Contrast Sets.*

As described in Section 2, STUCCO relies on categorical inputs. Inspired by STUCCO's tree search strategy, CCSM defines a novel discriminative pattern mining algorithm on continuous datasets. However, it is more likely that real world datasets contain mixed data types; Section 4 explains that crash data is high dimensional and heterogeneous. To enable comparisons between continuous and categorical contrast sets, we need a unified ranking algorithm that uses both types of features. The key is to make the anomaly scores for both types comparable.

To do so, we propose a new definition of anomaly score for categorical contrast sets. For a categorical contrast set X and group A, we want to see if the percentage (or support) of X is higher inside A than outside A. To achieve this, we use Cohen's h as the anomaly score, which is the difference between two proportions after an `arcsine` root transformation. Specifically, its formula is given by

$$h = 2(arcsin\sqrt{p_1} - arcsin\sqrt{p_2})$$

where p_1 and p_2 are the two sample proportions.

The goal of employing this transformation is to mitigate bias, where very rare contrast sets are disproportionately surfaced as significant and large. Without any transformation, the variance of the proportion is given by $p(1 - p)$; the variance is small when the proportion is close to 0.5 and large when it is close to 0 or 1. For example, if two proportions are both around 0.5, it is easy to detect their difference; if they are both close to 0, it is hard to detect their difference. The `arcsine` root transformation stabilizes the variance for all proportions, and hence, makes all proportion differences equally detectable. Cohen's d can be seen as standardized difference of means, while Cohen's h can be seen as difference of standardized proportions.

Note that Cohen's h has the same rule of thumb for categorizing the magnitude of the effect size as Cohen's d, making the two comparable to each other. Also note that the comparison of Cohen's d and two-sample t-statistic applies to Cohen's d and two-proportion z-statistic, which is another measure of the difference between two proportions, as well.

3.2.2 *Comparing anomaly score to percent difference in supports.*

The percent increase for a categorical contrast set to a group is defined as

$$\frac{\text{observed support} - \text{expected support}}{\text{expected support}}$$

Here, observed support is defined as the percentage of a contrast set in a certain group, whereas expected support is defined as the percentage of a contrast set across all the groups. By Taylor's Theorem, the new anomaly score (Cohen's h) for categorical contrast sets can be approximated by

$$\frac{\text{observed support} - \text{expected support}}{\sqrt{\text{expected support}(1 - \text{expected support})}}$$

up to some constant factor, under certain conditions. The major difference between these two definitions lies in the denominator, or how we standardize the difference between observed support and expected support. The new anomaly score not only comes with nicer statistical properties, but also mitigates the bias in ranking contrast sets using the old anomaly score. When a contrast set is very rare (i.e., expected support is very close to zero), the anomaly score of this contrast set, under the previous definition, will universally tend to be high. This makes ranking unfair to those comparatively frequent contrast sets (those with expected support not so close to zero).

However, under the new definition, when expected support is very close to zero, square root of expected support will cause it to deviate zero while (1 − expected support) stays very close to one. Thus it gives a larger denominator and the difference in the numerator is not over-standardized.

Moreover, the new anomaly score is closely connected to the Pearson's Chi Squared test for independence. Specifically, the χ^2 statistic is proportional to

$$\frac{(\text{observed support} - \text{expected support})^2}{\text{expected support}(1 - \text{expected support})}$$

which is simply the square of the approximation of Cohen's h displayed above. However, instead of using the Chi Squared test, we believe Cohen's h better suits our needs. The reason is that the anomaly score based on Cohen's h is positive only if observed support is larger than expected support, while the χ^2 statistic is always positive, even when observed support is smaller than expected support.

3.3 Confidence Intervals

For each anomaly we find, we provide confidence intervals for the mean and percentage difference to increase actionability of results.

The first confidence interval is provided for the effect size (Cohen's d or Cohen's h). For example, a Cohen's d of 0.5 tells us that based on the sample, there is a medium difference between current group and rest of the groups on a certain feature. If we further obtain a 95% confidence interval of (0.48, 0.52), we know that the sample is a good representation of the population, and we can be quite confident that there is a medium difference. However, if the interval is (0.19, 0.81), we are not certain that the difference is of a medium size; it could actually be large or small.

We use standard methods for constructing confidence intervals for Cohen's d and Cohen's h [5]. We apply Bonferroni correction on up to 20 anomalies for each group.

Another confidence interval is provided for the mean and percentage differences. We refer to (observed mean - expected mean) for continuous contrast sets and (observed percentage - expected percentage) for categorical contrast sets. This gives a better idea of what the difference looks like at the original scale without any standardization. For example, if a given feature has an expected support of 10% and an observed support of 50% in certain group, the difference of these two percentages is simply 40%; a confidence interval of (39%, 41%) would reassure developers that the percentage difference is estimated relatively precisely, compared to a confidence interval of (25%, 55%).

For continuous contrast sets, we use Welch's t-interval; for categorical contrast sets, we use Wilson score interval (without continuity correction). As before, we apply Bonferroni correction.

4 CONTRAST SET MINING IN PRACTICE

In this section, we give a broader picture of how CSM would be used in an industrial organization for crash diagnosis. Particularly, we describe the architecture of mobile app reliability tools at Facebook and how results from CSM can help developers.

4.1 Mobile App Reliability at Facebook

Figure 4 shows an overview of Facebook's mobile crash analysis architecture. When a client experiences a crash, the generated crash report is received by a "categorizer". The job of the categorizer is to assign the crash to a particular group of crashes, such that all crashes in the a group arise from the same root cause bug. The categorizer makes use of a mix of heuristics and ML-based approaches to compute these groups, and in the end produces a unique signature representing the group to which the crash was assigned. We will denote this group identifier as SIG and use it to refer to a group of crashes.

The crash report is also fed through a feature extractor that extracts metadata features such as the ones in Table 1. The features can then be processed by CSM algorithms – categorical features can be processed by the traditional STUCCO algorithm, and continuous features can be used as input to the CCSM algorithm presented in this paper. As shown in Figure 4, the SIGs that represent crash groups are used by CSM. The output of the CSM algorithms are contrast sets, which are conjunctions of feature-value pairs. While the categorizer acts upon the contents of crashes to compute the SIG, such as the stack trace and exception message, CSM algorithms use the metadata features to compute contrast sets. In this regard, CSM serves to explain properties of crash groups and why those crashes were grouped together.

Contrast sets are useful for a variety of downstream purposes. First, prominent contrast sets (based on our notion of anomaly score presented here) are useful for describing groups. Each SIG indexes into an internal issue tracker system where developers monitor spikes of crashes in the SIG, create tasks to work on bug fixes and mitigate issues. Contrast sets are ranked by their anomaly scores, which denotes the degree to which CCSM believes the contrast set to be prominent for the SIG, and displayed in the issue tracker UI. Since string keys assigned by categorizers do not have semantic meaning, contrast sets provide interpretable descriptions of groups of crash reports.

Second, they can surface spikes in SIGs that would otherwise go under the radar. For instance, users from a country Y using a particular build version X can be experiencing a spike of crashes. However, if the actual *number* of such crashes is small compared to the size of the SIG, the regression is unlikely to be surfaced at the SIG level. CSM would be able to produce the contrast set $\{\mathsf{country} : \mathsf{Y}, \mathsf{build\_version} : \mathsf{X}\}$, filtering on which would reveal the spike.

Finally, contrast sets can provide useful hints to developers debugging crashes in a SIG. Currently, the issue tracker UI displays a simple count of features, which, as we discussed in section 1, can be misleading. Contrast sets, on the other hand, surface features that statistically distinguish a SIG from others, guiding developers to which features are more likely to be related to the SIG.

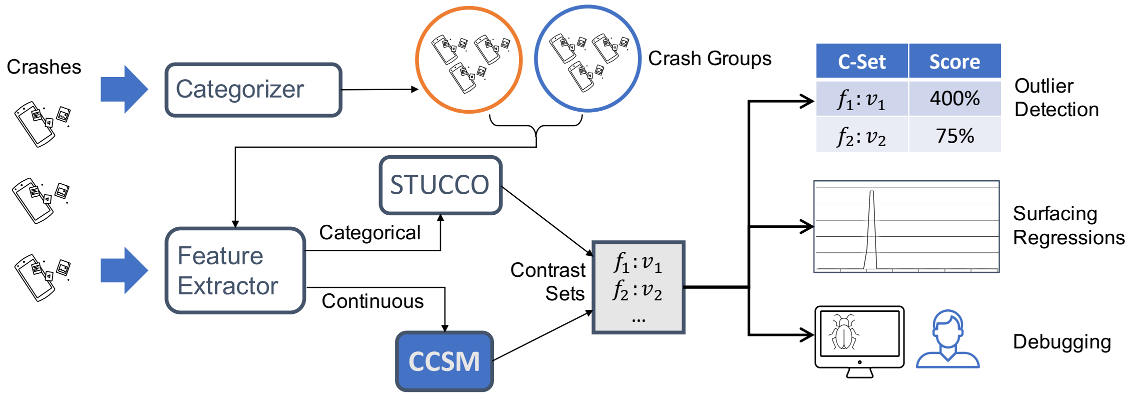


Figure 4: Overview of the crash analysis system

4.2 Usability of Contrast Sets

As noted by Webb *et al.* [11], visualizing contrast sets is a challenging open problem in the pattern mining space. We identified two key pain points to the consumption of contrast sets:

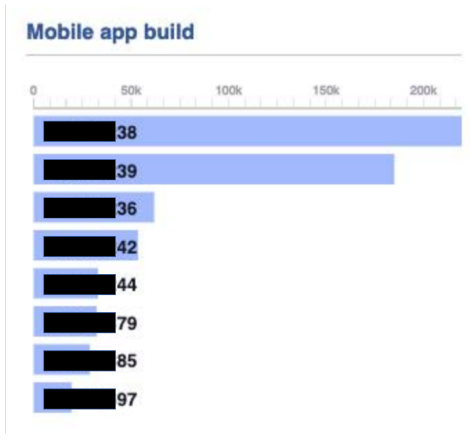
- *Actionability*: How much can we trust these findings? How likely are developers to act upon this information? Developers value transparency in ML models and want to quantify uncertainty. To address this, we provide a notion of *confidence* in Section 3 so that end users can assess the representativeness of our findings.
- *Interpretability*: Anomalous features should be immediately obvious to developers inspecting a group of crashes. This becomes a more challenging task when we consider conjunctive contrast sets as well.

To improve interpretability of statistical measures, we propose an alternative to the current visualization for crash statistics that only highlights meaningful statistical differences. As noted in Section 3.1, simple frequency counts can be misleading.

To illustrate the consequences of misleading visualizations, let us go through a realistic example. Figure 5(a) shows a ranking of mobile app builds for a particular SIG by *count*. Simply judging by the prevalence of build 38 in the SIG, one might incorrectly conclude that it is closely associated with the bug. CSM, on the other hand, revealed that build 38 is *expected* to be prevalent because it also occurred with similar frequencies in other SIGs, perhaps being the most used build of the app. Instead, it found that build 44 is the most anomalous as its anomaly score is 150% above expected thresholds, as shown in Figure 5(b). Engineers working on the SIG validated this insight, and eventually fixed the bug by gating out this build. This is an example of how CSM results can be presented to developers to aid them in debugging. In addition to UI tooling, we can integrate CSM results into the debugging workflow through scripts posting daily findings to oncall groups, and bots that automatically comment on open tasks associated with SIGs.

5 EVALUATION

To evaluate the proposed techniques, we consider the following questions:



(a)

Feature	Percent Deviation
{app_build: 44}	150%
{os_version: 4.0, country: US}	102%
{connection_type: WIFI}	45%
{device_brand: Samsung}	38%
...	

(b)

Figure 5: (a) Counts of app builds for a particular SIG (modified screenshot from the tool), (b) Anomalous features found by CSM

- **RQ1**: Does CCSM have lower computational cost than existing approaches? Is CCSM efficient enough to scale to high cardinality, high dimensional datasets?
- **RQ2**: Does contrast set mining help diagnose crash reports in our environment?
- **RQ3**: Does our definition of anomaly score add value over previous ranking techniques relying on percentage differences?

5.1 Implementation Details

To evaluate model efficiency, we collected on the order of $60k$ field crashes each day from the week of September 10, 2019 to September 16, 2019. The data consists of iOS Out-of-Memory (OOM) crashes from the core Facebook mobile app. As discussed in Section 2, OOM crashes are difficult bugs without accompanying stack traces, and are thus good candidates for contrast set mining. For each crash, we have metadata such as the device OS and app build as in Table 1, which include categorical, discrete, and continuous data types. Many crash reports include a sequence of navigation events, such as the example shown in Figure 2. We follow the procedure described in Section 2 to extract bigrams and embed them using TF-IDF weights, generating thousands of continuous columns for each dataset.

We simulate using CCSM at different points in time in a production setting. For our baseline, we use a standard implementation of STUCCO with equi-width binning. We compare CCSM to the binning approach with two different number of bins, 3 and 10. We demonstrate that discretization of bigrams is much slower than directly applying continuous contrast mining and generates lower quality contrast sets.

To mitigate the effects of uneven group sizes on our evaluation, we use stratified sampling to control the number of crashes fetched for each SIG in expectation. For each day, we ran our evaluation on $1k$, $10k$ and $60k$ crashes. At the end of each run, we recorded the runtime and inspected the output contrast sets to ensure quality of our results. We set an upper bound of $3600s$ (1 hour) for execution time; runs that exceeded this threshold were terminated due to high memory usage.

5.2 RQ1: Analysis of Execution Times

Figure 6 shows the execution times of different CSM models over time. The results vary over time because at each time interval, we collect a new set of crash reports. Since stratified sampling controls sample sizes in expectation, the cardinality of our dataset varies slightly as well across runs.

Both baseline approaches perform poorly when compared to CCSM. We find that discretization suffers from acute scalability problems. This is especially true of smaller bin widths; discretization with 10 bins consistently exceeded the time limit for the $N = 60k$ dataset. On average, CCSM achieves a $40x$ speedup over discretization with 10 bins and a $10x$ speedup over discretization with 3 bins. Prior work observed that using fewer bins generally leads to faster execution [13], but outputs fewer contrast sets and incurs greater information loss from bucketing. We validate these findings empirically in our results.

It should be noted that in pattern mining research, it is common to partition data ranges into hundreds of bins [13]. Since finer partitions would only further increase computational costs of discretization, we demonstrate our efficiency improvements on relatively simple baseline approaches.

5.3 RQ2: Validation of results

We considered 24 high priority crash tasks (all closed) generated from July-September 2019, where contrast mining generated findings. 16 of the tasks involved hard bugs, where stack traces were unavailable or difficult to parse. For these tasks, we selected 32 contrast sets we generated with the highest ranked anomaly scores. We have manually analyzed this set of crashes and the discussion and code changes that are attached to them, along with contrast mining findings. We labelled each contrast set as directly useful, relevant or compatible, and not helpful. We found 12 cases where the tool surfaced interesting patterns that were directly useful to the crash resolution; 18 cases where the tool generated compatible results but were not sufficient to root cause the bug, and 2 cases where our mining tool was not helpful.

5.4 RQ3: Improvement in Usability of Anomaly Scores

. In pattern mining we analyze the data in an exploratory fashion, where the emphasis is not on predictive accuracy but rather on finding previously unknown patterns in the data.

We thus use an example to illustrate how the new anomaly score and the current practice of using percent difference between expected and observed supports rank the features differently. We ran contrast set mining using data from July 2019. Figure 7 contains anomalies found for a regressing SIG, and the anomalies are ranked by the new anomaly score and the original percent difference based ranking.

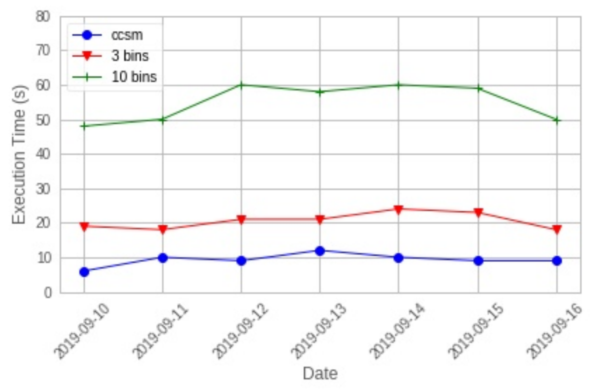
It is easy to see that the original anomaly score is in favor of finding anomalies with low expected support, such as `time_since_init_ms`: (0, 150000) and `background_time_since_init_ms`: (0, 1000).

The new anomaly score gives a higher rank to anomalies where the expected support is not extremely low, such as `major_app_version: 229`. This mitigates bias towards rare feature sets.

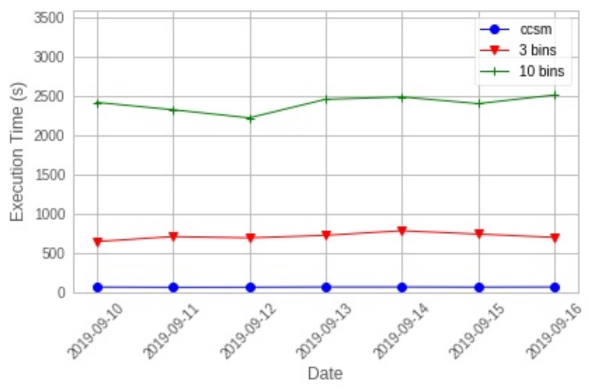
5.5 Early Experiences

Since experimenting with contrast set mining on field crashes, engineers trying the tool have found numerous cases where CCSM surfaced important insights on hard issues, and in some cases, found the root cause of groups of crashes. We find that the analysis of embedded navigation event sequences using CCSM adds significant value to crash analysis using STUCCO. By pinpointing specific frames, CCSM is able to provide more actionable insights than analysis on categorical variables alone. We describe several instances where both CCSM and STUCCO helped guide the debugging process below.

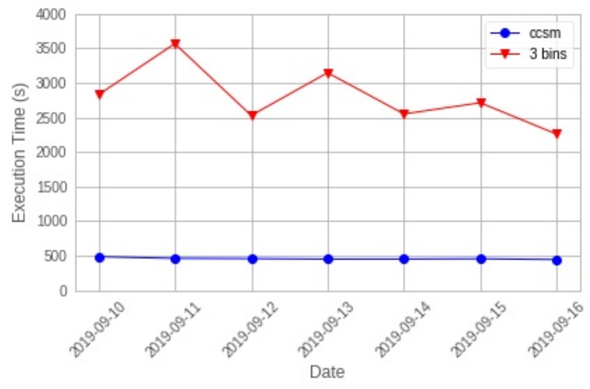
- *Root Cause for Hard Bugs.* Testing CCSM on production data over multiple days when a group of crashes was the most prevalent among users, engineers found that navigation to and from a navigation module related to comments showed up within the top five anomalous features consistently. Product engineers confirmed that the fix for the issue involved the navigation events surfaced by CCSM. This crash is an example of OOM errors, which are especially difficult to debug due to the lack of stack traces (see Section 2). This is a case where continuous CSM makes it possible to analyze hard bugs due to its scalability to high dimensional datasets. If STUCCO with equi-width binning instead of CCSM was applied, the anomalous navigation events might not be surfaced, since the limited sample size at the non-zero bins



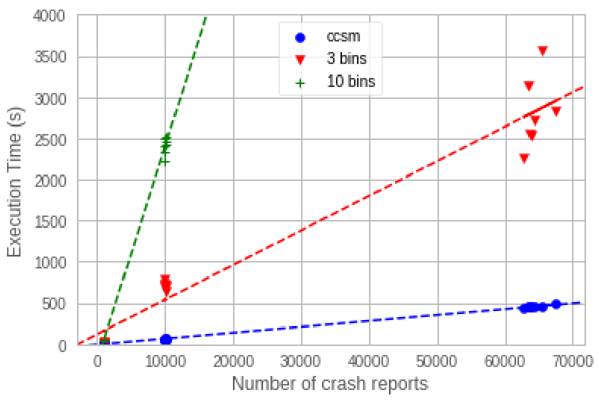
(a) $1k$ crash reports.



(b) $10k$ crash reports.



(c) $60k$ crash reports.



(a) Run time vs. Input size.

Figure 6: Execution times for different contrast mining implementations. The blue line is the CCSM approach.

Figure 7: Comparison of statistical ranking definitions

Features	Expected Support	Observed Support	Weighted Anomaly Score	Percent Difference
app version = 2	0.68	1	1.364	0.472
app build = 123	0.68	1	1.363	0.471
time since init = (0, 150000)	0.081	0.214	0.518	1.657
OS version = 12	0.742	0.858	0.355	0.156
background time since init = (0, 1000)	0.086	0.172	0.341	1.008

would be quite unlikely to yield a statistically significant difference, as discussed in Section 2.2.1.

- *Issue Discovery.* Engineers using the contrast mining tool saw that a certain app build was highly anomalous and experiencing high crash rates. For the corresponding SIG, contrast mining data showed that the number of app crashes we observed with this build was 56% higher than expected. This build number represents the x86 build type. Engineers confirmed that underlying issues such as this can otherwise be left unnoticed because the cohort of affected devices is very small, and signal is diluted because the build type failures are spread across 5-6 different SIGs.
- *Describing Crash Groups.* Contrast mining finds statistically significant deviations to help pinpoint the root cause of crashes. We ran contrast mining on a SIG associated with a high priority task. We then contacted the task owner with the list of anomalous features. The mobile engineer found the information very helpful as a way to differentiate between normal behavior and statistically significant differences. Specifically, one highly anomalous contrast set indicated that connection class was poor. "The fix that I proposed is based on the assumption that the network is slow, and this confirms that."

The qualitative case studies have been performed by authors to evaluate the algorithm in production setting with engineers and may suffer from selection bias as positive results are more likely to be reported.

6 RELATED WORK

There is a growing body of research applying machine learning techniques to crash triaging and resolution. Information retrieval based bug localization techniques extract semantic information from crash stacks, and have been shown to scale to large project sources with low cost text analysis [8]. Wu *et al.* located crash-inducing changes by training classification models on candidates extracted from buckets of crash reports [12].

The problem of bucketing crash reports has been well studied in literature. Dhaliwal *et al.* [6] found that crash clusters containing reports triggered by multiple bugs took longer to fix, and proposed grouping crashes using the Levenshtein distance between stack traces. Campbell *et al.* [3] found that off-the-shelf information retrieval techniques outperformed crash deduplication algorithms in categorizing crashes at scale. Fan *et al.* [7] conducted a comprehensive study of open source Java crash reports and distilled 11 common categories of developer errors.

For the most part, the above approaches focus on mining information at the trace level, or individual crash reports. Our approach focuses on analyzing characteristics of groups of crashes in aggregate to aid developers in crash triaging and resolution. Using our tool in conjunction with aforementioned crash analysis techniques can give developers more actionable insights, both for hard bugs and for standard crash reports with stack traces. Castelluccio *et al.* [4] presented the first application of CSM to the problem of crash group analysis. To the best of our knowledge, we are the first to bring contrast set mining to the continuous domain.

7 CONCLUSION AND FUTURE WORK

App crashes are severe symptoms of software errors, causing significant pain for both end users and oncall engineers. Maintaining app health is therefore one of the top priorities of large software organizations. We propose CCSM, a novel pattern mining algorithm for continuous data that scales to datasets with thousands of features. We found that automated crash analysis can detect anomalous patterns that are difficult to identify with manual inspection, and that the analysis of continuous features greatly adds value to existing categorical data mining approaches.

In the future, we plan to experiment with non-parametric methods to test statistical significance. Both the *one-way ANOVA* and *two-sample t-test* are known to be robust to the normality assumption. However, *one-way ANOVA* also assumes homogeneity of variances among the groups, which may not be realistic for distributions of field crashes. When this assumption is violated, we can use methods such as a *Kruskal-Wallis H Test* to test statistical significance instead.

Currently, CCSM also relies heavily on domain expertise to set user-defined thresholds to prune nodes that are not *large*. Exploring heterogeneous thresholds for different sets of features based on domain expertise will be useful contribution to the literature.

ACKNOWLEDGMENTS

We would like to thank Facebook engineers for their many engineering contributions and insight into unique challenges engineers face in a large software ecosystem. Saransh Bansal, Rushabh Shah and Phuong Nguyen applied contrast mining to Android reliability issues and provided domain expertise to help us design contrast mining tools. Alex Cohen and Jeremie Marguerie scoped out the continuous use case, and served critical roles in the application of CCSM to iOS reliability. Corrine Torrachio built visualizations of CCSM results. We also thank Eric Liaw, Surupa Biswas and Erik Meijer for their continued support for projects applying machine learning to software engineering problems.

REFERENCES

[1] Stephen D. Bay. 2000. Multivariate Discretization of Continuous Variables for Set Mining. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)*. ACM, New York, NY, USA, 315–319. https://doi.org/10.1145/347090.347159

[2] Stephen D. Bay and Michael J. Pazzani. 1999. Detecting Change in Categorical Data: Mining Contrast Sets. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '99)*. ACM, New York, NY, USA, 302–306. https://doi.org/10.1145/312129.312263

[3] Joshua Charles Campbell, Eddie Antonio Santos, and Abram Hindle. 2016. The Unreasonable Effectiveness of Traditional Information Retrieval in Crash Report Deduplication. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR '16)*. ACM, New York, NY, USA, 269–280. https://doi.org/10.1145/2901739.2901766

[4] Marco Castelluccio, Carlo Sansone, Luisa Verdoliva, and Giovanni Poggi. 2017. Automatically Analyzing Groups of Crashes for Finding Correlations. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*. ACM, New York, NY, USA, 717–726. https://doi.org/10.1145/3106237.3106306

[5] Jacob Cohen. 1992. Statistical Power Analysis. *Current Directions in Psychological Science* 1, 3 (1992), 98–101.

[6] Tejinder Dhaliwal, Foutse Khomh, and Ying Zou. 2011. Classifying field crash reports for fixing bugs: A case study of Mozilla Firefox. *IEEE International Conference on Software Maintenance, ICSM*, 333–342. https://doi.org/10.1109/ICSM.2011.6080800

[7] L. Fan, T. Su, S. Chen, G. Meng, Y. Liu, L. Xu, G. Pu, and Z. Su. 2018. Large-Scale Analysis of Framework-Specific Exceptions in Android Apps. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 408–419. https://doi.org/10.1145/3180155.3180222

[8] Shivani Rao and Avinash Kak. 2011. Retrieval from Software Libraries for Bug Localization: A Comparative Study of Generic and Composite Text Models. In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11)*. ACM, New York, NY, USA, 43–52. https://doi.org/10.1145/1985441.1985451

[9] Stephen Robertson. 2004. Understanding inverse document frequency: On theoretical arguments for IDF. *Journal of Documentation* 60 (2004).

[10] Mondelle Simeon and Robert Hilderman. 2008. Categorical Proportional Difference: A Feature Selection Method for Text Categorization. In *Proceedings of the 7th Australasian Data Mining Conference - Volume 87 (AusDM '08)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 201–208. http://dl.acm.org/citation.cfm?id=2449288.2449320

[11] Geoffrey I. Webb, Shane Butler, and Douglas Newlands. 2003. On Detecting Differences Between Groups. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03)*. ACM, New York, NY, USA, 256–265. https://doi.org/10.1145/956750.956781

[12] Rongxin Wu, Ming Wen, Shing-Chi Cheung, and Hongyu Zhang. 2018. Change-Locator: Locate Crash-inducing Changes Based on Crash Reports. In *Empirical Software Engineering 23 (ESE 2018)*. ACM, New York, NY, USA, 2866–2900. https://doi.org/10.1007/s10664-017-9567-4

[13] Gangyi Zhu, Yi Wang, and Gagan Agrawal. 2015. SciCSM: Novel Contrast Set Mining over Scientific Datasets Using Bitmap Indices. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management (SSDBM '15)*. ACM, New York, NY, USA, Article 38, 6 pages. https://doi.org/10.1145/2791347.2791361