The MAP Metric in Information Retrieval Fault Localization

Thomas Hirsch
Institute of Software Technology
Graz University of Technology
Graz, Austria
thirsch@ist.tugraz.at

Birgit Hofer
Institute of Software Technology
Graz University of Technology
Graz, Austria
bhofer@ist.tugraz.at

2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE) | 979-8-3503-2996-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ASE56229.2023.00041

***Abstract*—The MAP (Mean Average Precision) metric is one of the most popular performance metrics in the field of Information Retrieval Fault Localization (IRFL). However, there are problematic implementations of this MAP metric used in IRFL research. These implementations deviate from the text book definitions of MAP, rendering the metric sensitive to the truncation of retrieval results and inaccuracies and impurities of the used datasets. The application of such a deviating metric can lead to performance overestimation. This can pose a problem for comparability, transferability, and validity of IRFL performance results. In this paper, we discuss the definition and mathematical properties of MAP and common deviations and pitfalls in its implementation. We investigate and discuss the conditions enabling such overestimation: the truncation of retrieval results in combination with ground truths spanning multiple files and improper handling of undefined AP results. We demonstrate the overestimation effects using the Bench4BL benchmark and five well known IRFL techniques. Our results indicate that a flawed implementation of the MAP metric can lead to an overestimation of the IRFL performance, in extreme cases by up to 70 %. We argue for a strict adherence to the text book version of MAP with the extension of undefined AP values to be set to 0 for all IRFL experiments. We hope that this work will help to improve comparability and transferability in IRFL research.**

***Index Terms*—information retrieval, fault localization, MAP, mean average precision**

I. Introduction

Fault localization is a time-consuming and difficult part of the debugging process [1], [2]. To support developers in their work, researchers have proposed various approaches to automate this task [3]. While some of these approaches are test coverage based, others utilize execution traces, and again others use historical data and textual bug reports, e.g., Information Retrieval based Fault Localization (IRFL) approaches.

Typical IRFL approaches use bug reports as queries and source code repositories as the corresponding document libraries, and they apply information retrieval (IR) techniques to search within these libraries. The majority of IRFL approaches provide search results as a list of source code files ordered by suspiciousness [4]–[10]; only a few tools operate on method level [11], [12]. The inherent insensitivity of IRFL approaches to programming language, build system, and target platform significantly ease integration into existing software projects compared to test coverage based fault localization approaches. These advantages and promising performance results led to IRFL becoming an active research field.

Early research into IRFL applied various classic IR models and techniques, for example, Latent Dirichlet Allocation (LDA) [13]–[15], Latent Semantic Indexing [16], and Vector Space Models (VSM) [4], [17], [18]. VSM based approaches dominated the field for some time [5], [8], [11], but deep learning [6], [12], [19] and embedding based [20], [21] approaches have become more popular in recent years. Various extensions have been proposed to increase performance of IRFL by utilizing additional data sources and identifying and extracting structured data in both textual bug reports and source code. These include the utilization of historic bug reports [4], stack traces in bug reports [5], code structure [9], change histories [22], and combinations of them [8], [11].

To make the numerous approaches comparable, well established retrieval performance metrics from the wider field of IR are employed. The most popular metrics in IRFL are Mean Average Precision (*MAP*), Mean Reciprocal Rank (*MRR*), and *TopN* [23]. Researchers in the IR community have already discussed the characteristics of these retrieval performance metrics [24], [25] in depth, including discussions and evaluations on statistics around these metrics [26]. While researchers in the narrower field of IRFL have already investigated issues related to datasets [23], [27], evaluation methodologies [28], [29], and common assumptions in IRFL [30], [31], the limitations of the employed metrics and practical considerations have stayed untouched. To the best of our knowledge, there has been no investigation of the metrics' suitability for the field of IRFL, and no investigation into the practical application of these metrics in the context of IRFL research.

This paper focuses on the use of the popular *MAP* metric and the underlying *AP* metric, and its application in IRFL experiments. We discuss the origins and definitions of these metrics, and identify issues and problems with the *AP* metric in the field of IRFL, stemming from unclear definitions, implementation errors, and lack of standard implementations. These issues result in two versions of the *AP* metric that differ slightly in their implementation that are now in widespread use. We refer to these versions as AP_{mb} [24], [32] for the textbook version, and AP_{asrd} [33] for the discussed deviation. We show how truncated search results combined with ground

DOI 10.1109/ASE56229.2023.00041

truths spanning multiple items can impact the performance scores based on these metrics. In particular, AP_{asrd} is sensitive to such truncation, and overestimates the performance in such cases. Further, we show how improper handling of *undefined AP* scores and dataset inaccuracies and impurities can amplify the overestimation effect.

We demonstrate these effects on the Bench4BL [34] dataset, using five well known IRFL tools (BugLocator [4], BRTracer [5], AmaLgam [8], BLIA [11], BLUiR [9]). The application of AP_{asrd} on the Bench4BL dataset can spuriously inflate performance scores by up to 70 % in extreme cases. We show that the conditions that allow for such erroneous results due to improper metric implementation are common in Bench4BL. While we perform our demonstration using the Bench4BL dataset, we expect these conditions to be met in the majority of IRFL datasets/benchmarks [23].

This work aims to provide a clear picture of the *AP* and *MAP* metrics and to highlight issues and pitfalls to help researchers avoid them. Further, we want to raise awareness in the IRFL research community and the wider IR research community for the need of software libraries containing the default implementations of commonly used performance metrics, similar to what sklearn.metrics[1] provides for machine learning researchers and practitioners.

The contributions of this work are: In Section II, we discuss related work on IR study design, methodology, metrics, and statistical evaluations. We discuss the origins and different definitions of the *MAP* metric and its building blocks in detail in Section III. This is followed by a detailed problem statement in Section IV where we discuss the mathematical properties of the *AP* and *MAP* metrics. We then demonstrate the overestimation effects and the prevalence of the underlying conditions in IRFL research. In Section V, we describe our experimental setup including the chosen benchmark and research questions. In Section VI, we present our results, followed by our conclusion in Section VII.

II. Related Work

Various aspects of general IR experiments have been discussed in detail in the past, from study designs and methodologies, through suitability and applicability of performance metrics, to investigations on the suitability, sensitivity, and accuracy of various statistical tests:

Sutcliffe [35] explains study setups, experiment methodologies, evaluation strategies, performance measurement, model comparison, and statistical evaluation for general IR experiments in detail. The discussion on performance metrics for single queries envelopes recall, precision, fallout, and derivative metrics as the E measure [36] and MZ metric [37]. Various averaging strategies to create performance plots and single value performance summaries are discussed, however, *AP* and *MAP* are not mentioned.

Hull [38] discusses strategies for evaluating IR experiments using precision and recall based metrics. He investigates the properties of $P@k$ and $R@k$ metrics and precision-recall curves, and discusses the suitability of various statistical tests for these measures.

Buckley and Voorhees [24] investigate the expressiveness of common IR performance metrics for tool comparison. They discuss error rates and effect sizes of $P@k$ for various k, Recall after 1000 documents $R@1000$, Precision at 0.5 Recall, *R-Precision*, and *AP* performance metrics in conjunction with study size. They identified *AP* as comparably stable metric for retrieval performance comparison.

Cormack and Lyman [25] investigate statistical precision of *AP* and *MAP* metrics under variability of test datasets. They propose bootstrap methods to model corpus variability in order to predict confidence intervals for *AP* and *MAP*.

Sanderson and Zobel [26] discuss the sensitivity and reliability of various significance tests applied on performance results from IR experiments. They discuss reliability of *MAP* and $P@10$ under different conditions taking manual assessor effort into account, and conclude that t-test shows lower error rates as Wilcoxon and sign tests.

Smucker *et al.* [39] compare statistical significance tests applied on *MAP* scores for statistical evaluation of IR approaches. They conclude that Student's paired t-test, bootstrap, or Fisher's randomization tests perform well in IR experiments, while the use of Wilcoxon signed rank test and sign test suffer from false positives and false negatives and their use in IR research should be discontinued.

Kishida [40] discusses properties of *AP*, including sensitivity to changes in rankings and statistical reliability of the metric.

Most aspects of IR experiments discussed in the works above are valid and relevant in the more specialized field of IRFL. While—to the best of our knowledge—there are no separate investigations into IRFL specific study design, methodologies, performance metrics, and statistical evaluation, researchers have highlighted specific problems in IRFL experiments, mostly focused on the datasets used in such experiments. Tu *et al.* [28] highlight information leakage issues in a number of IRFL datasets and experiments leading to an overestimation of retrieval performance. Kochhar *et al.* [30] investigate potential biases in datasets used for IRFL experiments, showing that the inclusion of bug reports where the reporter already localizes the fault causes a bias and distorts retrieval performance results. Kim and Lee [23] point out issues regarding representability and generalizability of benchmarks used in IRFL. Hirsch and Hofer [27] survey public bug localization benchmarks and identify issues in data availability, data quality, and reproducibility. Kim and Lee [29] show how the inclusion of test files can impact the validity of IRFL studies. Wang *et al.* [31] show that the information necessary for IRFL to work effective is often not contained in bug reports. Further, they point out the lack of studies involving actual developers evaluating the usefulness IRFL approaches in practice.

Correctness of the utilized performance metrics' implementations is a general assumption in all of the above publications.

[1] https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

Our work differs from the above as it breaks with this assumption and discusses problematic implementations of the *AP* and *MAP* performance metric and investigates the resulting effects of performance results and comparisons.

III. Definitions

In this section, we discuss the *AP* and *MAP* metrics' origins and definitions in greater detail.

The *Documents* set is the unordered set of all available items in the document library. The *Relevant* set is an unordered set of relevant items for a given query, with $|Relevant|$ being its size. The *Retrieved* set is an ordered subset of *Documents*, with $|Retrieved|$ being its size. It contains the retrieved items for a given query in descending order of a similarity score, provided by the investigated IR tool. While the *Retrieved* set can list all items contained in the *Documents* set, in practice this list may be truncated, therefore containing only a subset of *Documents*. $|Relevant \cap Retrieved|$ is the number of relevant items in the list of retrieved items for a given query.

Recall R in a ranking problem is the portion of relevant documents that were retrieved. This is the number of relevant documents within the retrieved documents (true positives), divided by the number of relevant documents in total.

$$R = \frac{|Relevant \cap Retrieved|}{|Relevant|} \tag{1}$$

Precision P in a ranking problem is the portion of retrieved documents that are relevant. This is the number of relevant documents within the retrieved documents (true positives), divided by the number of retrieved documents (all positives).

$$P = \frac{|Relevant \cap Retrieved|}{|Retrieved|} \tag{2}$$

Precision at k $P@k$ is the precision at position k of the retrieved set, meaning the precision when cutting off the retrieved set at k.

$$rel(x) = \begin{cases} 1 & \text{if document at position } x \text{ is relevant} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$P@k = \frac{\sum_{i=1}^{k} rel(i)}{k} \tag{4}$$

Average Precision (*AP*)'s definitions vary slightly in notation and basis of calculation along two different styles. We discuss both definitions separately, and in detail, to show that they produce the same results.

Manning *et al.* define *AP* as "*the average of the precision values obtained for the set of top k documents existing after each relevant document*" [32]. Given a set of relevant documents $\{d_1, ..., d_n\} \subseteq Documents$ with $n = |Relevant|$ and $rank(d_i)$ being the position of the relevant document in the retrieved results, *AP* is calculated as follows:

$$AP_m = \frac{\sum_{i=1}^{|Relevant|} P@rank(d_i)}{|Relevant|} \tag{5}$$

Manning *et al.* explicitly state that relevant documents that are not part of the ranking have to be included in the calculation with a $P@k = 0$.

Arguably one of the most compact and concise definitions of AP can be found in Buckley and Voorhees [24], which states: "*Average Precision is the mean of the precision scores obtained after each relevant document is retrieved, using zero as the precision for relevant documents that are not retrieved.*" This formulation iterates over the retrieved documents ranking instead of iterating all relevant documents (as done by Manning *et al.* [32]) and is the most common form to be found in IRFL literature:

$$AP_b = \frac{\sum_{k=1}^{|Retrieved|} P@k \cdot rel(k)}{|Relevant|} \tag{6}$$

AP_m and AP_b produce identical results, although they slightly differ in notation. We will refer to this metric as AP_{mb} from now on, referring to it as the correct textbook implementation of this metric.

However, in case of an empty relevant document set, $|Relevant| = 0$, both Manning *et al.*'s [32] and Buckley and Voorhees' [24] AP are *undefined* due to a division by zero. While neither of them discusses this possibility, we argue that AP should be set to 0 in such scenarios.

AP_{mb} is not to be confused with *average precision at seen relevant documents* as defined in the first edition of Yates and Ribeiro-Neto's book "Modern Information Retrieval" [33]. Yates and Ribeiro-Neto assume a ranking that is not exhausting the full document library, being truncated at some point, and the ranking including only a portion of the relevant documents. *Average precision at seen relevant documents* is then calculated as the sum of $P@k$ of the relevant documents in the ranking, divided by the number of relevant documents occurring in the ranking:

$$AP_{asrd} = \frac{\sum_{k=1}^{|Retrieved|} P@k \cdot rel(k)}{|Relevant \cap Retrieved|} \tag{7}$$

Yates and Ribeiro-Neto [33] remark that a high *average precision at seen relevant documents* can still have poor performance in terms of recall. This metric can be used in precision oriented settings, and in cases where an exhaustive ground truth is unavailable, therefore the number of *Relevant* documents is (yet) unknown [41]. We will refer to this metric as AP_{asrd} in this work.

Finally, *Mean average precision MAP* is the mean of all queries' $q \in Q$ average precision:

$$MAP = \frac{1}{|Q|} \sum_{q=1}^{|Q|} AP(q) \tag{8}$$

MAP provides a single value score summarizing the performance of a retrieval system over a number of queries. *MAP* is commonly defined using AP_{mb} [24], [32], [41]. While it is possible to use AP_{asrd}, such use in the context of incomplete ground truth is known to be problematic [41].

Along with *MRR* and *TopN*, *MAP* is one of the most popular performance metrics used in IRFL research [23]. The *Retrieved* documents in the context of IRFL experiments is the set of bug fix locations in the codebase, most often presented at file level, and commonly referred to as *ground truth* in IRFL. A common assumption in fault localization experiments is that a complete ground truth is available. This is reflected in the benchmarks and datasets used in such experiments [23], [27]. In this work we consider the ground truth to be a set of source code files.

IV. Problem Statement

Some definitions and implementations of the *AP* metric used in IRFL research do not conform with the textbook version AP_{mb}. This results in incompatible *MAP* metrics voiding any comparison between publications, or even comparison between tools within the same publication. First, we discuss the mathematical properties of the textbook definitions of AP_{mb}, *MAP*, and deviating implementations in detail. We examine the conditions that have to be met for performance overestimations to occur. Finally, we briefly discuss the usage of the *AP* and *MAP* metrics in IRFL research papers, and the prevalence of the underlying conditions in IRFL experiments.

A. AP versions dissection

While the *Retrieved* set can possibly list all items from the document library, in practice it is often truncated to some degree, therefore containing only a subset of the full document library. The most common divergence, and the one that we focus on in this work, stems from different handling of relevant but unretrieved documents in the calculation of *AP*, manifesting in different values used for X in the following generic definition of AP_x in Eq. 9.

$$AP_X = \frac{\sum_{k=1}^{|Retrieved|} P@k \cdot rel(k)}{X} \tag{9}$$

Setting $X = |Relevant|$ will produce an AP_{mb} according to Manning *et al.* [32] and Buckley and Voorhees [24] acknowledging unretrieved relevant documents with a $P@k = 0$, while setting $X = |Relevant \cap Retrieved|$ will produce an AP_{asrd} according to Yates and Ribeiro-Neto [33] simply ignoring unretrieved relevant documents.

This leads to AP_{asrd} and AP_{mb} producing different results under certain conditions, that we will now discuss in detail: If a retrieval operation returns the full document library D, therefore producing an exhaustive ranking, all relevant documents will be retrieved and therefore both metrics produce the same result as Eq. 10 applies.

$$\begin{aligned} Retrieved = D &\implies |Relevant \cap Retrieved| = |Relevant| \\ &\implies AP_{asrd} = AP_{mb} \end{aligned} \tag{10}$$

Further, if only a single document d_1 is in *Relevant*, both metrics will produce the same result as long as *Retrieved* is truncated after the relevant document, as shown in Eq. 11.

$$\begin{aligned} |Relevant| = 1 \land |Retrieved| \geq rank(d_1) \\ \implies AP_{asrd} = AP_{mb} \end{aligned} \tag{11}$$

If *Retrieved* is truncated before the occurrence of the highest ranked document d_1, AP_{mb} will result 0 while AP_{asrd} will be undefined due to division by 0, as shown in Eq. 12.

$$\begin{aligned} |Retrieved| < rank(d_1) &\implies |Relevant \cap Retrieved| = 0 \\ &\implies AP_{mb} = 0 \\ &\implies AP_{asrd} = undef. \end{aligned} \tag{12}$$

If the ground truth is empty, both metrics will be undefined.

$$\begin{aligned} |Relevant| = 0 &\implies AP_{mb} = undef. \\ &\implies AP_{asrd} = undef. \end{aligned} \tag{13}$$

However, if there are multiple relevant documents, and the *Retrieved* documents are truncated, it is possible that not all relevant documents are retrieved. In such a case AP_{asrd} and AP_{mb} could differ in X in Eq. 9 and therefore possibly produce different results.

$$\begin{aligned} |Relevant \cap Retrieved| \leq |Relevant| \\ \implies AP_{mb} \leq AP_{asrd} \end{aligned} \tag{14}$$

If *Retrieved* are truncated in a way that no relevant documents are removed, AP_{asrd} will be equal to AP_{ab}. However, if one or more relevant documents are removed AP_{asrd} will be strictly greater than AP_{mb} unless there are no relevant documents left.

$$\begin{aligned} |Retrieved| \geq max(rank(Relevant)) &\implies AP_{asrd} = AP_{mb} \\ |Retrieved| < max(rank(Relevant)) &\implies AP_{asrd} > AP_{mb} \end{aligned} \tag{15}$$

Further, as relevant documents are truncated from *Retrieved* documents, AP_{asrd} and AP_{mb} will deviate in different directions. AP_{mb} will decrease as a before non-zero $P@k$ in Eq. 9 will become zero, while AP_{asrd} will increase as divisor X in Eq. 9 representing $|Relevant \cap Retrieved|$ will decrease. Given multiple relevant documents, a smaller cutoff, i.e. stronger truncation, will produce lower values for AP_{mb} and higher values for AP_{asrd}, therefore increasing the difference between the two metrics.

$$\begin{aligned} &Retrieved_{@(k)} = cutoff(Retrieved, k) \\ &Retrieved_{@(k-1)} = cutoff(Retrieved, k-1) \\ &\implies |Retrieved_{@(k)}| > |Retrieved_{@(k-1)}| \\ &\implies AP_{mb@(k)} \geq AP_{mb@(k-1)} \\ &\implies AP_{asrd@(k)} \leq AP_{asrd@(k-1)} \\ &\implies AP_{asrd@(k)} - AP_{mb@(k)} \leq AP_{asrd@(k-1)} - AP_{mb@(k-1)} \end{aligned} \tag{16}$$

Table I shows an exemplary ranking performed on a document library with size $|D| = 15$, and highlights the difference in calculation and results of AP_{mb} and AP_{asrd} when truncating the rankings to $|Retrieved| = 10$.

The discussion above describes the mechanism behind AP_{asrd} overestimating performance using the example of a ranking being truncated at different levels. While the use of AP_{asrd} obviously poses a problem for comparing tools that produce different output ranking lengths, favoring the tool producing shorter rankings, this issue also impacts expressiveness and validity of comparison when both tools are

TABLE I: AP_{mb} and AP_{asrd} for an exemplary ranking to be cut off at $k = 10$.

$\|Retrieved\| = \|D\| = 15$			$\|Retrieved\| = k = 10$		
Rank	Relevant	P@K	Rank	Relevant	P@K
1	1	1	1	1	1
2	1	1	2	1	1
3			3		
4			4		
5	1	0.6	5	1	0.6
6			6		
7			7		
8			8		
9			9		
10			10		
11					
12	1	0.33		-	0
13					
14					
15	1	0.33		-	0
AP_{mb}	$X = 5$	0.65	AP_{mb}@10	$X = 5$	0.52
AP_{asrd}	$X = 5$	0.65	AP_{asrd}@10	$X = 3$	**0.87**

TABLE II: AP_{mb}@10 and AP_{asrd}@10 for two different tools' rankings.

Tool A			Tool B		
Rank	Relevant	P@K	Rank	Relevant	P@K
1	1	1	1	1	1
2			2		
3			3		
4	1	0.5	4		
5			5		
6			6		
7			7		
8			8		
9			9		
10			10		
11			11		
12			12	1	0
13			13		
14			14		
15			15		
AP_{mb}@10	$X = 2$	0.75	AP_{mb}@10	$X = 2$	0.5
AP_{asrd}@10	$X = 2$	0.75	AP_{asrd}@10	$X = 1$	1.0

truncated to the same length. Table II shows two exemplary rankings produced by hypothetical tools A and B, and their corresponding performance scores $AP_{asrd@10}$ and $AP_{mb@10}$ at cutoff $k = 10$. We can observe that while Tool B clearly produces the inferior rankings, its $AP_{asrd@10}$ performance is higher than that of Tool A.

Finally, the handling of queries producing undefined AP_{asrd} due to division by 0 as shown in Eq. 12 can additionally impact *MAP* scores. Removal of such undefined datapoints instead of setting AP_{asrd} to 0 equals a reduction of the number of queries $|Q|$ used in the divisor for the calculation of *MAP*. This results in MAP_{asrd} further overestimating MAP_{mb}.

$$|Q| \geq |Q_{drop\_undef}| \implies MAP_{asrd} \geq MAP_{mb} \tag{17}$$

B. AP in IRFL research

The conditions necessary for such an inflation of *AP*/*MAP* scores are found in various research papers and corresponding implementations. The ground truth to span multiple files is a standard assumption in IRFL, and very common to be found in IRFL datasets and benchmarks [4], [34], [42], [43]. The prevalence of this assumption is the main reason for the usage of the *MAP* metric, considering all relevant items in a ranking, contrasted by *RR* and *TopN* metrics that only consider the single highest ranking relevant item.

In a preliminary investigation, we found that some real world IRFL tools do in fact truncate their results, producing various lengths of ranking outputs. For example, BLIA [11] returns a comparably small subset of the document corpus, while BugLocator [4] and BRTracer [5] return exhaustive rankings containing the whole document library.

Whether a tool produces an exhaustive ranking, or if only a subset may be returned, is not discussed in any of the papers we reviewed. Such truncations and exclusions may stem from error handling, may be a design choice, or have other technical reasons. However, uncovering such behavior by performing code reviews is difficult and time consuming. Inspecting actual tool outputs is the most promising path to determine if, and to what degree, a tool truncates the output rankings.

Most of the reviewed publications describe *AP* / *MAP* as defined by Buckley and Voorhees [24] (see Eq. 6), e.g., Wang and Lo [7], [8] (AmaLgam), Zhou *et al.* [4] (BugLocator), Lee *et al.* [34] (Bench4BL), Saha *et al.* [9] (BLUiR), and Takahashi *et al.* [44]. However, some researchers redefine *AP* in their papers to align with Yates and Ribeiro-Neto's [33] AP_{asrd} metric, e.g., Wong *et al.* [5] (BRTracer) and Rahman and Roy [10].

Beyond these, there are other definitions of the metric that further deviate from the textbook version AP_{mb} to be found in IRFL research papers [23], [45], [46] and definitions that are unclear to whether $X = |Relevant \cap Retrieved|$ or $X = |Relevant|$ was used [11], [21], [47].

When investigating the source code and datasets published alongside some publications, we found that the implementations may differ from the definitions in the corresponding papers. For example, Takahashi *et al.* [44] define *AP* along Buckley and Voorhees [24], but their implementation uses $X = |Relevant \cap Retrieved|$ in Eq. 6, or Kim and Lee [23] define *AP* to sum $P@k$ not only the relevant positions but at all positions, while correctly implementing *AP* according to Eq. 6. Others strictly oblige to the textbook definitions in both their publications and implementations, as for example, Wang and Lo [7], [8] (AmaLgam) use the total number of relevant items according to ground truth.

Incorrect ground truths in IRFL datasets, as documented by Kim and Lee [23], can further widen the gap between AP_{mb} and AP_{asrd} scores. For example, in a preliminary investigation we found the ground truths provided in the Bench4BL dataset [34] to be inflated with files that were added in commits after the provided code base snapshots, rendering them unretrievable. Such *bloated ground truths* will decrease AP_{mb} scores, further widening the gap between the two metrics.

To summarize, there are mainly two versions of the *AP* metric used in IRFL research. The first is the textbook definition

of *AP* by Manning *et al.* [32] and Buckley and Voorhees [24], the second is similar to Yates and Ribeiro-Neto's [33] *average precision at seen relevant documents* which is problematic due to its insensibility to recall. It is difficult—and sometimes impossible—to determine which version of the metric was employed. Results calculated with a given metric may differ based on the level of truncation of a tool's outputs. If a tool's produced rankings are truncated, and to what degree, is not discussed in any of the reviewed papers, and only determinable via experiment. Thus, *AP* and *MAP* values obtained from different tools and evaluations may not be directly comparable, even when they are based on the same datasets.

V. Experiment Setup

We want to demonstrate the magnitude and prevalence of errors that may arise from applying deviating implementations of the *AP* metric to measure fault localization performance on an established IRFL bug benchmark, namely Bench4BL [34]. In this section, we first explain the reasons for choosing Bench4BL and the benchmark's structure and issues, before we present our research questions.

A. Benchmark

A number of benchmarks/datasets specifically designed for the evaluation of IRFL approaches are publicly available [27]. Our selection criteria are dataset size, integration of existing IRFL tools, and age of the benchmark. We have selected Bench4BL [34] for our evaluation because of its size, completeness of the provided data, it being a rather young and contemporary dataset, and its integration of five existing IRFL tools. The benchmark also contains the AspectJ, SWT, ZXing, JDT, and PDE datasets already used in previous IRFL research [4], [5], [11], [48], [49]. The included tools are BugLocator [4], BLIA [11], [47], BLUiR [9], AmaLgam [7], [8], and BRTracer [5]. Further, the Bench4BL authors provide a scripting framework for automating setup and execution of experiments. The included scripts were used as-is for running the IRFL tools to produce the rankings used in our experiments.

Contents and provided materials: Bench4BL [34] consists of 10 017 bugs from 51 different open source Java projects from five different organziations (Apache, Spring, Wildfly, Commons, JBoss). These bugs are distributed over 695 different versions of these software projects. Bench4BL contains the code base of a project for each of these different versions to be used as the document corpus for bug reports created on this version.

Part of the Bench4BL benchmark is a smaller collection of bug datasets that have been in widespread use in previous IRFL research. This collection spans 558 bugs from AspectJ, JDT, PDE, SWT, and ZXing, and is discussed under the label of "*old subjects*" in the original publication. The AspectJ bugs originate from the widely used iBugs dataset [48]. We will use this "*old subjects*" sub-dataset only in RQ1, as its structure slightly differs from the remainder of Bench4BL, resulting in added complexity and compatibility issues for the remainder of our evaluation. The following dissemination of the Bench4BL benchmark focuses only on its "*new subjects*".

TABLE III: BugLocator MAP_{mb} and MAP_{asrd} on various cleaning stages of Bench4BL.

Dataset	MAP_{mb}	MAP_{asrd}	Overestimation
Unmodified Bench4BL	0.4006	0.4153	3.663 %
Rectified bloated ground truth	0.4159	0.4153	-0.137 %
Rectified ambiguous filenames	0.4158	0.4160	0.048 %

The benchmark does not provide intermediate files as, for example, the outputs of the included IRFL tools. We therefore reran the Bench4BL experiments for BugLocator, BRTracer, and BLIA ourselves. We only consider valid rankings produced by these tools. For example, non-existent ranking outputs due to crashes, empty output files, and other obvious erroneous outputs are excluded from our evaluation. BugLocator and BRTracer produced 9278 valid rankings, BLIA produced 8709 valid rankings.

Due to dependencies to outdated Java libraries, rerunning the Bench4BL experiments for BLUiR and AmaLgam is not a straight forward task. We therefore use the intermediate files and tool outputs provided by Takahashi *et al.* [44] for BLUiR and AmaLgam. Takahashi *et al.* [44] have removed program versions that contain less than five bugs in their work, the resulting number of rankings available to us is 6931 for both BLUiR and AmaLgam.

Dataset Issues: In preliminary experiments we discovered some issues with the benchmark that influence and distort the calculated AP_{asrd} and AP_{mb} scores. We calculated the MAP_{ab} and MAP_{asrd} scores of BugLocator on the full Bench4BL benchmark. We selected BugLocator for this discussion as it returns exhaustive rankings covering all existing files in the target project's repository. MAP_{ab} and MAP_{asrd} should in theory provide the same results in this scenario.

However, MAP_{asrd} is 3.66 % higher than MAP_{mb} as shown in Table III. When we investigated this effect, we found that there are files listed in Bench4BL's ground truth that do not exist in the given code base. We will refer to this effect as *bloated ground truth*.

Bench4BL provides the code base of the target projects at certain tagged versions. The bugs associated with a specific version have their bug fixing commits an unknown number of commits after the tagged version commit. A file that is added in a commit after the version tagged commit, but before the bug fixing commit, can therefore occur in the change set of the bug fix, but not exist in the provided version of the code base. About 19% of bugs in Bench4BL have such a *bloated ground truth*. A benchmark that would avoid such behavior would require document corpora based on the commit just before the fix was applied. In order to rectify, we remove files from the ground truth that do not exist in the target projects' source repositories.

When removing these files, MAP_{asrd} is now 0.14 % lower than MAP_{mb} (see Table III), which should not be possible according to their definitions. Investigation showed that this

effect arises from ambiguous file names in the rankings and ground truth. Such ambiguity arises from somewhat unconventional formatting for file identifiers produced by BugLocator and other tools. The first part of the path being the Java package, followed by the classname and a '.java' file ending. However, such naming schema can only uniquely identify a file in a source code repository as long as there is only a single *src* folder containing all Java code. Although rare, such name collisions occur in the Bench4BL projects, for example, if there are multiple Java projects with a similar structure located in the same repository. If such an ambiguous file identifier occurs in the ground truth, it will lead to multiple files in the tools' ranking to be marked as relevant, and therefore increase the number of relevant files used for the calculation of MAP_{asrd}. This constellation occurs in approximately 0.3% of bugs when using BugLocator rankings. We will refer to this effect as *ambigous file names*.

In order to rectify, we keep only the first occurrence of a file identifier in a tool's ranking, removing later duplicated occurrences. The corresponding MAP_{asrd} and MAP_{mb} scores including this final cleaning step are shown in Table III in row 'Rectified ambiguous filenames'.

We found that some of the investigated tools can produce file names that miss parts or complete path information and therefore cannot be matched with the ground truth, or the projects' source for that matter. Examples are '$.DirectComponent.java' and '__TOP_LEVEL_PACKAGE__. __SEGMENT_PACKAGE__.ScaffoldMobileApp-template.java'. These entries are often superfluous, simply inflating the rankings' lengths, while sometimes the corresponding correct file identifiers are missing from the rankings. Such behavior can be observed for multiple IRFL tools, most notably BugLocator and BRTracer with the above described behavior, and AmaLgam by default not returning full paths, but only the top five Java package levels. We cannot rectify these other issues.

In the following experiments we will either refer to the *cleaned* Bench4BL dataset, where both *bloated ground truth* and *ambigous file names* issues were rectified, or the *unmodified* Bench4BL dataset.

B. Research questions

As established in Section IV, two basic conditions have to be met to enable AP_{asrd} producing higher scores than AP_{mb}: (1) the ground truth contains more than a single file, and (2) the results of the tool are truncated / are only a subset of the full document library. To quantify these two conditions, we have the following two research questions:

RQ1: How big is the average ground truth in Bench4BL datasets, and what proportion of bugs have a ground truth containing multiple files?

RQ2: Do the IRFL tools included in Bench4BL truncate their results?

To answer RQ1 and RQ2, we analyze the data provided by the Bench4BL datasets, and the corresponding outputs of IRFL tools when executed on this dataset.

Then, we investigate the magnitude of overestimation of AP_{asrd} over AP_{mb} as a function of truncation level of otherwise identical rankings:

RQ3: How strong is AP_{asrd} overestimating AP_{mb} for truncated BugLocator retrieval results on the Bench4BL dataset? To answer RQ3, we truncate the rankings at various lengths and calculate the AP_{asrd} and AP_{mb} scores. We use the ranking results produced by BugLocator executed on Bench4BL as the basis for this analysis. We have chosen BugLocator as it produces untruncated, exhaustive rankings and as the tool has become a de-facto standard baseline for IRFL performance comparisons [5], [8], [9], [11], [50].

Within RQ3, we also investigate the combined effects of truncation and incorrect ground truths and the impact of removal of *undefined AP* scores:

RQ3a: How strong is AP_{asrd} overestimating AP_{mb} for truncated BugLocator retrieval results when considering the bloated ground truth issue found in Bench4BL?

RQ3b: How strong is AP_{asrd} overestimating AP_{mb} for truncated BugLocator retrieval results when undefined AP values are simply ignored?

VI. Results and Discussion

In the first part of this section, we evaluate and discuss RQ1 and RQ2 on the basis of the Bench4BL dataset and its included IRFL tools. We then evaluate and discuss RQ3 on the basis of BugLocator rankings from a cleaned Bench4BL dataset. In RQ3a and RQ3b we go into further detail, evaluating and discussing the additional and amplifying effects of dataset and ground truth issues, and different strategies of dealing with undefined *AP* scores.

A. RQ1: Ground truth size

To answer RQ1, we analyze the ground truth provided for each bug in the Bench4BL dataset. The *bloated ground truth* issue discussed in Section V-A directly influences this measure, and we therefore calculate this for both, the unmodified Bench4BL, and for the cleaned Bench4BL dataset.

In the unmodified Bench4BL dataset, the average number of ground truth files per bug is 3.48, with 43.75 % of bugs having more than one file in its ground truth. In the cleaned dataset, the average number of ground truth files per bug is 2.68, with 38.25 % of bugs having more than one file in its ground truth. The average ground truth size of Bench4BL's "*old subjects*" sub-dataset is 3.30 files per bug, with 59.13 % of bugs having multiple files in their ground truths.

These results, together with the widespread use of Bench4BL, especially its "*old subjects*" sub-datasets, show that ground truths covering multiple files are a quite common occurrence in IRFL experiments.

B. RQ2: Tool truncation behavior

To answer RQ2, we analyze the outputs of BugLocator, BRTracer, BLIA, BLUiR, and AmaLgam running on Bench4BL bugs. We count the number of Java files in the versions' codebase using cloc[2] version 1.92 to establish a document

[2]https://github.com/AlDanial/cloc

corpus size for each bug. We then measure the length of the ranking produced by these tools for each bug.

Table IV shows the number of bugs/rankings, the per bug average document corpus size for each bug, the per bug average ranking length, and exhaustiveness of a tool's rankings as the average percentage of ranking length against number of document corpus length. We do not apply any data cleaning, and do not rectify any of the dataset issues discussed in Section V-A. Superfluous file identifiers in BugLocator's rankings result in their length being slightly above 100 % of the total number of available files. BLIA shortens output rankings to about 12 % of the number of java files available in the underlying projects' repositories. BLUiR and AmaLgam rankings are 7 % shorter than the number of files available. We cannot determine if this behavior is the result of a conscious design choice, result of error and exception handling within the tool, or by accident, due to a lack of documentation.

TABLE IV: Averages of number of existing files $|files|$, number of $|Retrieved|$ files, and coverage of the tools ranking in % on the unmodified "*new subjects*" Bench4BL dataset.

Tool	# Bugs	$\|files\|$	$\|Retrieved\|$	coverage in %
BugLocator	9278	2911	2920	100.4
BRTracer	9278	2911	2905	99.8
BLIA	8709	3020	353	11.9
BLUiR	6931	2658	2509	92.9
AmaLgam	6931	2658	2505	92.8

C. RQ3: AP sensibility on truncation

To investigate the difference in *AP* and *MAP* resulting from the different versions' sensibility on truncation level rankings, we use the *cleaned* version (see Section V-A) of "*new subjects*" Bench4BL dataset. We run BugLocator on the benchmark yielding 9278 rankings for bug reports. We implement AP_{mb} using the number of relevant items based on the provided ground truth, and AP_{asrd} using the number of relevant items in the ranking. If the calculated scores are *undefined* due to division by zero, we set the result to 0. Based on these two *AP* versions, we calculated resulting MAP_{mb} and MAP_{asrd} values, truncating BugLocator rankings to fractions of the document library length. Please keep in mind that the underlying rankings stay the same, only differing in their length, being truncated after the first k results.

Table V shows the resulting scores and the overestimation of MAP_{asrd} over MAP_{mb} in percent. The effect of overestimation diminishes towards untruncated rankings as expected. Truncating the rankings to 10 % of the document library length can produce a MAP_{asrd} that is 5.3 % higher than the MAP_{mb}. At a rather shallow truncation level of 90 % MAP_{asrd} is still overestimating by 0.24 %.

Figure 1 shows MAP_{mb} and MAP_{asrd} against the fractions of the document library length k that the rankings were truncated at. MAP_{mb} of the un-truncated ranking is used as a baseline. MAP_{mb} slightly underestimates the performance if rankings are truncated. This is explained by unretrieved documents impacting AP_{mb} with their $P@k$ value of 0 in this case. However, MAP_{asrd} will overestimate performance of truncated rankings, as low recall is not punished.

TABLE V: MAP_{mb} and MAP_{asrd} scores of BugLocator evaluated on the cleaned "*new subjects*" Bench4BL dataset, truncating results at fractions of the document library length.

Fraction	MAP_{mb}	MAP_{asrd}	Overestimation in %
0.1	0.413	0.435	5.35
0.2	0.415	0.428	3.19
0.3	0.415	0.424	2.15
0.4	0.415	0.422	1.60
0.5	0.416	0.421	1.20
0.6	0.416	0.419	0.86
0.7	0.416	0.418	0.63
0.8	0.416	0.418	0.44
0.9	0.416	0.417	0.24
1.0	0.416	0.416	0.05

This overestimation trend continues as retrieval results are further truncated. The strongest truncation that we identified in IRFL experiments [10], [51], [52] occurs in the form of a *MAP*@10 score, using only the top $k = 10$ items of retrieval results. Calculating such *MAP*@10 score using AP_{asrd} results in performance being overestimated by 18.53 %.

D. RQ3a: AP sensibility on Dataset Issues

We repeated our experiment, calculating MAP_{mb} and MAP_{asrd} on the *unmodified* version of Bench4BL, to investigate the combined effects of ranking truncation on an imperfect dataset. Figure 2 illustrates that the gap between reported AP_{mb} and AP_{asrd} performance scores is widened by dataset inaccuracies as discussed in Section V-A. The inaccuracies and issues in the Bench4BL dataset, especially the *bloated ground truth*, lead to a correctly implemented AP_{mb} score underestimating performance, while AP_{asrd} stays mostly unaffected, as can be observed by comparing Figure 2 and Figure 1. The resulting MAP_{mb} and MAP_{asrd} scores for a 0.1 fraction of the rankings are 0.398 and 0.435, and for untruncated rankings the corresponding scores are 0.401 and 0.415. These results demonstrate how dataset inaccuracies and impurities as for example a *bloated ground truth* further amplify the overestimating effect of AP_{asrd}.

E. RQ3b: Effect of undefined AP

As already mentioned in Section IV, *AP* can be *undefined* due to division by zero. For AP_{mb}, this occurs only when cleaning the *bloated ground truth* results in an empty ground truth. However, when applying AP_{asrd}, dividing by the number of relevant items contained in the retrieved items, such occurrences are now connected to the level of truncation. The more the results are truncated, the higher the probability that all relevant items are removed from the retrieved items, and therefore AP_{asrd} being *undefined*.

We investigate the effect size on *MAP* if this behavior is ignored and data points where *AP* evaluates to *undefined* are simply dropped from the evaluation. To do so, we repeat the basic experiment outlined in RQ3 and truncate BugLocator

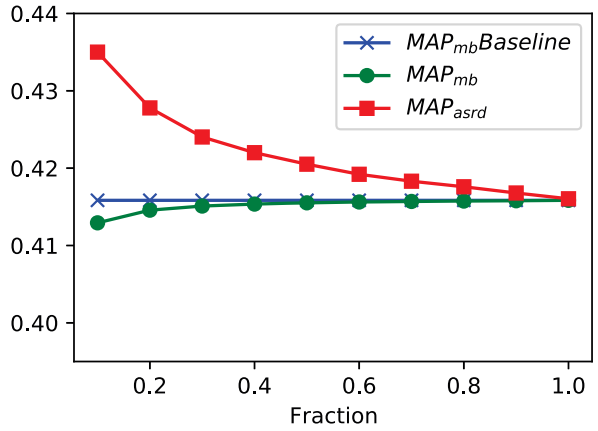

Fig. 1: MAP_{mb} and MAP_{asrd} of BugLocator rankings truncated at fractions of the document library length. Un-truncated MAP_{mb} is shown as a baseline. The cleaned "*new subjects*" Bench4BL dataset was used.

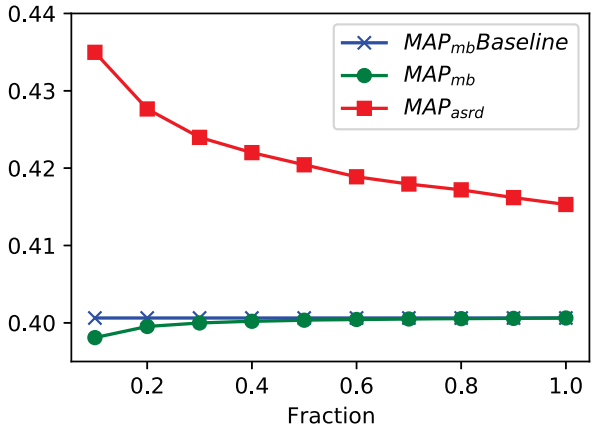

Fig. 2: MAP_{mb} and MAP_{asrd} of BugLocator rankings truncated at fractions of the document library length. Un-truncated MAP_{mb} is shown as a baseline. The unmodified "*new subjects*" Bench4BL dataset was used.

retrieval results to fractions of the corresponding document library size. However, this time, we simply remove data points where $AP_{asrd} = undefined$ and $AP_{mb} = undefined$. We use the cleaned version of Bench4BL dataset to isolate the effect.

Figure 3 shows BugLocator's *MAP* values based on AP_{mb}, AP_{asrd}, for both strategies, dropping *undefined* data points, and setting them to zero, for the uncleaned dataset. Dropping data points leads to even stronger overestimation of *MAP* performance calculated from AP_{asrd}. Truncating the retrieval results to 10 % of the document corpus size leads to a spurious *MAP* performance increase of 17.26 % when comparing AP_{asrd}*drop-undef* against textbook AP_{mb}. Truncating retrieval results to only the top $k = 10$ items of retrieval results to calculate a *MAP*@10 score using AP_{asrd}*drop-undef* results in 55.63 % overestimation of the correct AP_{mb} based score.

While the removal of the *bloated ground truth* results in some bugs having empty ground truths, impacting MAP_{mb}*drop-undef* scores, this is not the case for the unmodified Bench4BL with its non-empty ground truths. Removing *undefined AP* values from the evaluation on the *unmodified* Bench4BL dataset further widens the gap between the two metrics. MAP_{mb}*drop-undef* equals MAP_{mb} in this case as no empty ground truths occur, while AP_{asrd}*drop-undef* is further inflated. Truncating the retrieval results to 10 % of the document corpus size leads to a spurious *MAP* performance increase of 21.60 % when comparing AP_{asrd}*drop-undef* against textbook AP_{mb}. Truncating retrieval results to only the top $k = 10$ items of retrieval results to calculate a *MAP*@10 score using AP_{asrd}*drop-undef* results in 71.59 % overestimation of the correct AP_{mb} based score.

We therefore argue that *AP* should be set to 0 if the ground truth is empty, to avoid spurious increases in *MAP* performance due to possible errors in ground truth, mapping of documents and retrieval results, and deviating metric implementations, e.g., AP_{asrd}.

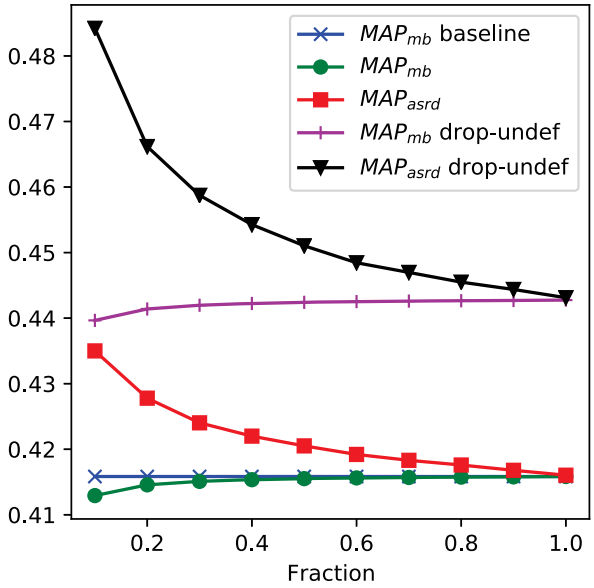

Fig. 3: MAP_{mb} and MAP_{asrd} of BugLocator rankings truncated at fractions of the document library length. Un-truncated MAP_{mb} is shown as a baseline. The cleaned "*new subjects*" Bench4BL dataset was used. Undefined instances of *AP* are either dropped or set to 0.

F. Statistical significance

The overestimation effect is not stochastic in nature, but mathematical defined by $AP_{asrd} \geq AP_{mb}$ and $AP_{asrd@(k-1)} \geq AP_{asrd@(k)}$ for cutoff k as discussed in our problem statement in Section IV. This effect will only occur for a discrete number of bugs in the dataset for any given cutoff, with the remainder reporting equal scores. However, as a single data

point with inflated AP_{asrd} within a sample will be reflected as an increase in corresponding *MAP* score, $MAP_{asrd} > MAP_{mb}$ and $MAP_{asrd@(k-1)} > MAP_{asrd@(k)}$ applies for any practical considerations. As this change is strictly one directional, statistical tests performed on the resulting distributions will report a statistical significance bordering on certainty for any reasonable sample size.

This is most apparent and easiest to demonstrate using paired sign based tests, as for example, the sign test and Wilcoxon signed rank test, as the signs of all unequal scores point strictly towards $AP_{asrd@k}$ and $MAP_{asrd@k}$ with shorter cutoffs k by definition. While sign based tests have been found to have inferior expressiveness for IR experiments [26], [39] than paired Students t-test, they are still used in IRFL research [5], [8], [44]. However, we argue that other statistical tests used in IRFL experiments (e.g., Wilcoxon rank-sum and paired Students t-test) will also report high significance when applied to *MAP* scores, favoring $MAP_{asrd@k}$ with shorter cutoffs k, and favoring MAP_{asrd} over MAP_{mb}.

To summarize, while we demonstrated and discussed the effect size in detail, we refrain from performing statistical significance tests as the underlying problem is not stochastic in nature and these tests will report high significance by default. However, we want to point out that statistical significance tests are not capable of uncovering such spurious performance improvements caused by deviating metric implementations.

G. Implications

Assume a hypothetical IRFL tool extending BugLocator, or any other tool for that matter. This hypothetical tool applies BugLocator and truncates the produced rankings to a cutoff k without performing any other modifications. Using $MAP_{asrd@k}$ as performance metric, we can "improve" performance scores by reducing the cutoff length k. Using Bench4BL as evaluation dataset, performance scores can be spuriously improved almost arbitrarily by modifying k, while common statistical significance tests will report a high statistic significance of the improvement. We want to highlight that this is not a dataset/benchmarking issue, nor is there an issue with truncated rankings per se, but this is an inherent issue of the MAP_{asrd} metric. Utilizing the textbook definitions AP_{mb} and MAP_{mb}, reducing the cutoff length k, performance scores would be moderately penalized instead.

H. Threats to validity

The main threat to external validity is the selection of Bench4BL for our demonstration of effect sizes. We limit this threat by analyzing the underlying conditions enabling the reported overestimation, most notably the ground truth spanning multiple files. We show that this condition is also met in the "*old subjects*" sub-dataset in Bench4BL that has been used in numerous IRFL experiments in the past. Further, we perform an in-depth analysis of the Bench4BL dataset to identify and discuss the contained noise, inaccuracies, and other issues. We implement steps to clean the dataset wherever possible to reduce the influence on our experiment setup. We discuss the effects of such dataset issues on the investigated performance scores in separation.

We argue that a ground truth spanning multiple files is a common assumption in IRFL research, and the widespread use of *MAP* being a result of this, as this metric provides a single value summary for rankings containing more than a single relevant file. Further, we mathematically discussed and dissected the different versions of the *MAP* metric to show the underlying problems. Based on this analysis, we are confident to argue that the problem discussed in this paper will impact any IRFL experiments on any dataset, if the ground truths span multiple files and retrieval results are truncated.

VII. Conclusion

In this work, we have discussed how deviation from the textbook definition of the *AP* metric can lead to overestimation of *AP* and *MAP* performance scores. We have examined the mathematical properties of the textbook version of the AP metric (AP_{mb}) and one common implementational deviation (AP_{asrd}) and have discussed the conditions that can lead to erroneous performance results: When the ground truth contains more than one file, AP_{asrd} is sensitive to truncation of retrieval results and overestimates the performance compared to the textbook version AP_{mb}.

To underline our mathematical analysis, we have demonstrated the relevance and magnitude of the problem on the Bench4BL dataset, a benchmark specifically designed for evaluating and comparing different IRFL approaches: Between 38 % and 44 % of bugs contained in Bench4BL have a ground truth spanning multiple files. In the "*old subjects*" subset of Bench4BL, this ratio is even higher with 59 % of bugs listing multiple files as ground truth. This supports our argument that ground truths spanning multiple files is not only a common assumption in IRFL research, but is also reflected in the corresponding IRFL datasets and experiments.

Out of the five investigated IRFL tools (BugLocator, BRTracer, BLIA, BLUiR, AmaLgam), only BugLocator and BRTracer produce untruncated, exhaustive retrieval results. The truncation levels of the remaining tools range widely from containing about 90 % of all available files in their ranking, to truncating results to only about 10 % of the document library.

Truncating BugLocator's retrieval results to 10 % of the document library size results in MAP_{asrd} overestimating MAP_{mb} by more than 5 % on the cleaned dataset, and more than 9 % on the unmodified Bench4BL dataset. The $MAP@10$ performance of BugLocator on a cleaned Bench4BL is overestimated by more than 18 % when using $MAP_{asrd}@10$ instead of the textbook implementation of $MAP_{mb}@10$.

Incorrect handling of empty ground truth sets—that can occur when truncating rankings— further inflate MAP_{asrd} scores. Retrieval results truncated to 10 % size, in combination with a simple removal strategy of *undefined* AP_{asrd} scores result in MAP_{asrd} overestimating performance by more than 21 % over the textbook implementation of MAP_{mb}. Such removal strategy applied to the calculation of $MAP_{asrd}@10$ leads to a 72 % overestimation.

To summarize, when applied to truncated retrieval results, AP_{mb} slightly underestimates performance, while AP_{asrd} overestimates performance. Impurities, inaccuracies, and errors in datasets amplify this effect. Incorrect handling of *undefined* *AP* scores further widen the gap.

This creates a problem for transferability and comparability of results across different research papers in the field of IRFL. We advise to exclusively use *AP*/*MAP* according to the definitions in Manning *et al.* [32] for iterating *relevant* items (see Eq. 18) and Buckley and Voorhees [24] for iterating *retrieved* items (see Eq. 19), with the extension of using $AP = 0$ in *undefined* points, in IRFL research to avoid these problems.

$$AP = \begin{cases} \frac{\sum_{i=1}^{|Relevant|} P@rank(d_i)}{|Relevant|} & \text{if } |Relevant| > 0 \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

$$AP = \begin{cases} \frac{\sum_{k=1}^{|Retrieved|} P@k \cdot rel(k)}{|Relevant|} & \text{if } |Relevant| > 0 \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

Further, we ask researchers to describe their tools' truncation behavior in their papers. Finally, we would like to call out to the wider IR community for the creation of standard software libraries containing implementations of the most common performance metrics.

VIII. Data avalability

The evaluation scripts and dataset artifacts used in our experiments are publicly available on Zenodo [53].

IX. Acknowledgment

The work described in this paper has been funded by the Austrian Science Fund (FWF): P 32653-N (Automated Debugging in Use).

References

[1] B. Beizer, *Software Testing Techniques*, 2nd ed. Itp Media, 1990.

[2] T. Hirsch and B. Hofer, "What we can learn from how programmers debug their code," in *8th International Workshop on Software Engineering Research and Industrial Practice (SER-IP)*, 2021, pp. 37–40.

[3] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A Survey on Software Fault Localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, aug 2016.

[4] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports," in *International Conference on Software Engineering (ICSE)*, 2012, pp. 14–24.

[5] C. P. Wong, Y. Xiong, H. Zhang, D. Hao, L. Zhang, and H. Mei, "Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis," in *30th International Conference on Software Maintenance and Evolution (ICSME)*, dec 2014, pp. 181–190.

[6] A. N. Lam, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Bug localization with combination of deep learning and information retrieval," in *IEEE International Conference on Program Comprehension (ICPC)*, jun 2017, pp. 218–229.

[7] S. Wang and D. Lo, "Version history, similar report, and structure: Putting them together for improved bug localization," in *22nd International Conference on Program Comprehension (ICPC)*, 2014, pp. 53–63.

[8] ——, "AmaLgam+: Composing rich information sources for accurate bug localization," *Journal of Software: Evolution and Process*, vol. 28, no. 10, pp. 921–942, oct 2016.

[9] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2013, pp. 345–355.

[10] M. M. Rahman and C. K. Roy, "Improving IR-based bug localization with context-aware query reformulation," in *26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 621–632.

[11] K. Youm, J. Ahn, and E. Lee, "Improved bug localization based on code change histories and bug reports," *Information and Software Technology*, vol. 82, pp. 177–192, 2017.

[12] X. Li, W. Li, Y. Zhang, and L. Zhang, "DeepFL: Integrating multiple fault diagnosis dimensions for deep fault localization," in *28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, jul 2019, pp. 284–295.

[13] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Source code retrieval for bug localization using latent Dirichlet allocation," in *Working Conference on Reverse Engineering (WCRE)*, 2008, pp. 155–164.

[14] ——, "Bug localization using latent Dirichlet allocation," *Information and Software Technology*, vol. 52, no. 9, pp. 972–990, sep 2010.

[15] A. T. Nguyen, T. T. Nguyen, J. Al-Kofahi, H. V. Nguyen, and T. N. Nguyen, "A topic-based approach for narrowing the search space of buggy files from a bug report," *26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 263–272, 2011.

[16] P. Shao, T. Atkison, N. Kraft, and R. Smith, "Combining lexical and structural information for static bug localisation," *International Journal of Computer Applications in Technology*, vol. 44, no. 1, pp. 61–71, 2012.

[17] S. Rao, H. Medeiros, and A. Kak, "An incremental update framework for efficient retrieval from software libraries for bug localization," in *Working Conference on Reverse Engineering (WCRE)*, 2013, pp. 62–71.

[18] C. Tantithamthavorn, R. Teekavanich, A. Ihara, and K.-I. Matsumoto, "Mining a change history to quickly identify bug locations: A case study of the Eclipse project," in *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2013, pp. 108–113.

[19] A. Lam, A. Nguyen, H. Nguyen, and T. Nguyen, "Combining deep learning with information retrieval to localize buggy files for bug reports," in *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 476–481.

[20] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu, "From word embeddings to document similarities for improved information retrieval in software engineering," in *38th International Conference on Software Engineering (ICSE)*, may 2016, pp. 404–415.

[21] N. Miryeganeh, S. Hashtroudi, and H. Hemmati, "GloBug: Using global data in fault localization," *Journal of Systems and Software*, vol. 177, p. 110961, 2021.

[22] B. Sisman and A. Kak, "Incorporating version histories in information retrieval based bug localization," in *IEEE International Working Conference on Mining Software Repositories*, 2012, pp. 50–59.

[23] M. Kim and E. Lee, "Are datasets for information retrieval-based bug localization techniques trustworthy?: Impact analysis of bug types on IRBL," *Empirical Software Engineering*, vol. 26, no. 3, pp. 1–66, may 2021.

[24] C. Buckley and E. M. Voorhees, "Evaluating evaluation measure stability," *ACM SIGIR Forum*, vol. 51, no. 2, pp. 235–242, aug 2017.

[25] G. V. Cormack and T. R. Lynam, "Statistical precision of information retrieval evaluation," *29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, vol. 2006, pp. 533–540, 2006.

[26] M. Sanderson and J. Zobel, "Information retrieval system evaluation: Effort, sensitivity, and reliability," in *28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2005, pp. 162–169.

[27] T. Hirsch and B. Hofer, "A systematic literature review on benchmarks for evaluating debugging approaches," *Journal of Systems and Software*, vol. 192, oct 2022.

[28] F. Tu, J. Zhu, Q. Zheng, and M. Zhou, "Be careful of when: An empirical study on time-related misuse of issue tracking data," *26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, vol. 18, pp. 307–318, oct 2018.

[29] M. Kim and E. Lee, "Are information retrieval-based bug localization techniques trustworthy?" in *International Conference on Software Engineering (ICSE)*. ACM, may 2018, pp. 248–249.

[30] P. S. Kochhar, Y. Tian, and D. Lo, "Potential biases in bug localization: Do they matter?" in *29th ACM/IEEE International Conference on Automated Software Engineering (ASE)*, 2014, pp. 803–813.

[31] Q. Wang, C. Parnin, and A. Orso, "Evaluating the usefulness of IR-based fault localization techniques," in *International Symposium on Software Testing and Analysis*. ACM, 2015.

[32] C. Manning, P. Raghavan, and H. Schütze, *Introduction to modern information retrieval (2nd edition)*. Cambridge University Press, 2008.

[33] R. Baeza-Yates and B. Ribeiro-Neto, *Modern information retrieval*. Addison Wesley, 1999.

[34] J. Lee, D. Kim, T. F. Bissyandé, W. Jung, and Y. Le Traon, "Bench4BL: Reproducibility study on the performance of IR-based bug localization," in *27th International Symposium on Software Testing and Analysis (ISSTA)*, jul 2018, pp. 61–72.

[35] J. Tague-Sutcliffe, "The pragmatics of information retrieval experimentation, revisited," *Elsevier*, 1992.

[36] C. J. Van Rijsbergen, "Foundation of evaluation," *Journal of Documentation*, vol. 30, no. 4, pp. 365–373, 1974.

[37] W. M. Shaw, "On the foundation of evaluation," *Journal of the American Society for Information Science*, vol. 37, no. 5, pp. 346–348, 1986.

[38] D. Hull, "Using statistical testing in the evaluation of retrieval experiments," *Annual International ACM SIGIR Conference on Research and Development in Infofmation Retrieval*, pp. 329–338, 1993.

[39] M. D. Smucker, J. Allan, and B. Carterette, "A comparison of statistical significance tests for information retrieval evaluation," in *International Conference on Information and Knowledge Management (CIKM'07)*, 2007, pp. 623–632.

[40] K. Kishida, "Property of average precision and its generalization: An examination of evaluation indicator for information retrieval experiments," *NII Technical Reports*, vol. 2005, no. 14, pp. 1–19, 2005.

[41] N. Craswell and S. Robertson, *Average precision at n, in Encyclopedia of Database Systems*. Boston, MA: Springer US, 2009, pp. 193–194.

[42] R. K. Saha, Y. Lyu, W. Lam, H. Yoshida, and M. R. Prasad, "Bugs.jar: A large-scale, diverse dataset of real-world Java bugs," in *15th International Conference on Mining Software Repositories (MSR)*, 2018, pp. 10–13.

[43] M. Garnier and A. Garcia, "On the evaluation of structured information retrieval-based bug localization on 20 C# projects," in *30th Brazilian Symposium on Software Engineering (SBES)*, sep 2016, pp. 123–132.

[44] A. Takahashi, N. Sae-Lim, S. Hayashi, and M. Saeki, "An extensive study on smell-aware bug localization," *Journal of Systems and Software*, vol. 178, aug 2021.

[45] B. Wang, L. Xu, M. Yan, C. Liu, and L. Liu, "Multi-dimension convolutional neural network for bug localization," *IEEE Transactions on Services Computing*, vol. 15, no. 3, pp. 1649–1663, 2022.

[46] A. Chen, T. Chen, and S. Wang, "Pathidea: Improving information retrieval-based bug localization by re-constructing execution paths using logs," *IEEE Transactions on Software Engineering*, 2021.

[47] K. C. Youm, J. Ahn, J. Kim, and E. Lee, "Bug localization based on code change histories and bug reports," in *Asia-Pacific Software Engineering Conference (APSEC)*, may 2016, pp. 190–197.

[48] V. Dallmeier and T. Zimmermann, "Extraction of bug localization benchmarks from history," in *ACM/IEEE International Conference on Automated Software Engineering (ASE'07)*, 2007, pp. 433–436.

[49] S. Rao and A. Kak, "Retrieval from software libraries for bug localization: A comparative study of generic and composite text models," in *International Conference on Software Engineering*, 2011, pp. 43–52.

[50] X. Ye, R. Bunescu, and C. Liu, "Learning to rank relevant files for bug reports using domain knowledge," in *ACM SIGSOFT Symposium on the Foundations of Software Engineering*, nov 2014, pp. 689–699.

[51] M. Kim and E. Lee, "A novel approach to automatic query reformulation for IR-based bug localization," in *34th ACM/SIGAPP Symposium on Applied Computing (SAC '19)*, 2019, pp. 1752–1759.

[52] Y. Yang, Z. Wang, Z. Chen, and B. Xu, "Context-aware program simplification to improve information retrieval-based bug localization," *22nd International Conference on Software Quality, Reliability and Security (QRS)*, pp. 252–263, dec 2022.

[53] T. Hirsch and B. Hofer, "Supplementary material for 'The MAP metric in Information Retrieval Fault Localization'," apr 2023. [Online]. Available: https://zenodo.org/record/7817016