Parameterizing and Assembling IR-based Solutions for SE Tasks using Genetic Algorithms

Annibale Panichella[1], Bogdan Dit[2], Rocco Oliveto[3],
Massimiliano Di Penta[4], Denys Poshyvanyk[5], Andrea De Lucia[6]
[1] Delft University of Technology, The Netherlands – [2] Boise State University, USA
[3] University of Molise, Pesche (IS), Italy – [4] University of Sannio, Benevento, Italy
[5] The College of William and Mary, VA, USA – [6] University of Salerno, Fisciano (SA), Italy

***Abstract*—Information Retrieval (IR) approaches are nowadays used to support various software engineering tasks, such as feature location, traceability link recovery, clone detection, or refactoring. However, previous studies showed that inadequate instantiation of an IR technique and underlying process could significantly affect the performance of such approaches in terms of precision and recall. This paper proposes the use of Genetic Algorithms (GAs) to automatically configure and assemble an IR process for software engineering tasks. The approach (named GA-IR) determines the (near) optimal solution to be used for each stage of the IR process, *i.e.,* term extraction, stop word removal, stemming, indexing and an IR algebraic method calibration. We applied GA-IR on two different software engineering tasks, namely traceability link recovery and identification of duplicate bug reports. The results of the study indicate that GA-IR outperforms approaches previously published in the literature, and that it does not significantly differ from an ideal upper bound that could be achieved by a supervised and combinatorial approach.**

***Keywords*—Text-based software engineering, Search-based software engineering, Information Retrieval, Parametrization**

I. Introduction

Prior research in software engineering (SE) highlighted the usefulness of conceptual (or textual) unstructured information to capture the knowledge and design decisions of software developers. Identifiers and comments encoded in class or method names, or attributes in source code and other artifacts contain information often indispensable for program understanding [2], [6], [13], [23], [29], [46]. This conceptual information plays a paramount role as a data source, which is used by many (semi-) automatic techniques to support software maintenance and evolution tasks.

In recent years, many researchers concentrated on the problem of extracting, representing, and analyzing conceptual information in software artifacts. Specifically, Information Retrieval (IR) methods were proposed and used to support practical tasks. Early approaches aimed at constructing software libraries [32] and supporting reuse tasks [35], [51], while more recent work focused on addressing software maintenance tasks including feature location (*e.g.,* [41]), traceability link recovery (*e.g.,* [1], [34]), change impact analysis (*e.g.,* [19]), identification of duplicate bug reports (*e.g.,* [48], [50]).

All these IR-based techniques that support SE tasks, such as Latent Semantic Indexing (LSI) [12] or Latent Dirichlet Allocation (LDA) [4], require configuring different components and their respective parameters, such as the type of pre-processors (*e.g.,* splitting compound or expanding abbreviated identifiers; removing stop words and/or comments), stemmers (*e.g.,* Porter or snowball), indexing schema (*e.g.,* term frequency - inverse document frequency, tf-idf), similarity computation mechanisms (*e.g.,* cosine, dot product, or entropy-based). Nevertheless, despite this overwhelming popularity of IR methods in SE research, most of the proposed approaches are based on ad-hoc guidelines and methods for choosing, applying, assembling, and configuring IR techniques.

Recent research has shown that different calibration of the IR processes can produce varied results in the context of duplicate requirements identification [18] or fault localization [49]. Also, some IR techniques, such as LDA, require a proper calibration of their parameters [21], [38]. Moreover, *recent studies demonstrate that the effectiveness of these IR-based approaches not only depends on the design choices behind the IR techniques and their internals, but also on the type of software artifacts used in the specific SE tasks and more generally on the project datasets* [37], [38]. Many existing SE approaches using IR methods rely on ad-hoc methods for configuring these components and solutions, oftentimes resulting in suboptimal performance of such promising analysis methods to deal with unstructured software data. This challenges the practical use of IR-based approaches and undermines the technology transfer to software industry.

In this paper we propose **GA-IR**, an approach that aims at enabling automatic search and assembly of parameterized IR-based solutions for a given software engineering task and a dataset that takes into account not only task specific components and data sources (*i.e.,* different parts of software artifacts related to solving a particular SE task), but also internal properties of the IR model built from the underlying dataset using a large number of possible components and configurations. We use Genetic Algorithms (GAs) to effectively explore the search space of possible combinations of instances of IR constituent components (*e.g.,* pre-processors, stemmers, indexing schemas, similarity computation mechanisms) to select the candidates with the best expected performance for a given dataset used for a SE task. During the GA evolution, the quality of a solution (represented as a GA individual) is evaluated based on the quality of the clustering of the indexed software artifacts, following a process similar to what was previously done for calibrating LDA parameters [38]. For this reason—and differently from previously proposed approaches for IR-process configuration [31]—our approach has the following noteworthy advantages:

1) GA-IR is *unsupervised*, hence it does not require a manually labeled training set, oftentimes unavailable, *e.g.,* when one needs to recover traceability links for a project for the first time;
2) GA-IR is *task independent*, *i.e.,* given the available dataset, it assembles and configures an IR technique that produces a high-quality clustering, which results in an improved performance for a specific task.

We empirically show that using *GA-IR* it is possible to automatically assemble a near-optimal configuration of an IR-based solution for datasets related to two kinds of software engineering tasks: (i) traceability link recovery and (ii) duplicate bug report identification. The evaluation shows that IR techniques instantiated by *GA-IR* outperform previously published results related to the same tasks and the same datasets, and that the performances of *GA-IR* do not significantly differ from an ideal upper bound, obtained by means of a combinatorial search, and with the availability of an oracle (which is not required by our approach).

Structure of the paper. Section II provides background information on IR-based software engineering. Section III describes the proposed GA-IR approach. Section IV describes the study that we performed to evaluate the GA-IR. The results are reported in Section V, while the threats to validity are discussed in Section VI. Section VII discusses related literature, while Section VIII concludes the paper and outlines directions for future work.

II. Background

This section provides some background on a generic IR process for solving SE problems, and how such a process can be instantiated for solving two problems that we believe are relevant IR applications to SE tasks, namely traceability link recovery and identification of duplicate bug reports.

A. A generic Information Retrieval based approach

Let us consider a generic IR-based approach, as the one shown in Fig. 1. Next, we describe the details behind each of the steps comprising such an approach.

Step 1: Term extraction. This step consists of removing elements (*e.g.,* special characters) that are not relevant to the IR process, and extracting portions of software artifacts that are considered relevant for the task at hand. For example, when performing requirements-to-source-code traceability recovery, one may (not) decide to consider source code comments [1], may (not) decide to split compound identifiers [14], or may decide to consider only certain parts-of-speech, *e.g.,* nouns [5]; similarly, when comparing bug reports for duplicate detection, contributor comments' (other than the initial description), stack traces, and source code fragments may (not) be considered.

Step 2: Stop word removal. This step aims at removing common terms that often do not contribute to discerning one document from another. This can be done using a stop-word list, *e.g.,* by removing stop words (*e.g.,* articles, prepositions, commonly used verbs), programming language keywords, or recurring domain-specific terms.

Step 3: Morphological analysis. This step is often performed to bring back words to the same root (*e.g.,* by removing plurals

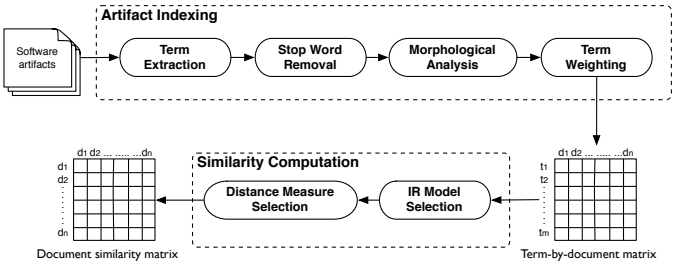


Fig. 1. Outline of a generic IR technique for solving SE problems.

to nouns, or verb conjugations). The simplest way to perform morphological analysis is to rely on stemming algorithms, *e.g.,* the Porter stemmer [39].

Step 4: Term weighting. The information extracted in the previous phase is stored in an $m \times n$ matrix, called *term-by-document matrix* (*TDM*), where m is the number of terms occurring in all the artifacts, and n is the number of artifacts in the repository. A generic entry $w_{i,j}$ of this matrix denotes a measure of the weight (*i.e.,* relevance) of the i^{th} term in the j^{th} document [3]. Different measures of relevance can be used: the simplest one is the *Boolean* weighting, which just indicates whether a term appears in a document or not; other measures are the term frequency, which accounts for the *term frequency* (*tf*) in a document, or the *tf-idf* (*term frequency-inverse document frequency*), which gives more importance to words having high frequency in a document (high *tf*) and appearing in a small number of documents, thus having a high discriminating power (high *idf*). In general, one can use a combination of a local weight of the term in a specific document (*e.g.,tf*) and a global weight of the term in the whole document collection (*e.g.,idf*). A more sophisticated weighting schema is represented by *tf-entropy* [17], where the local weight is represented by the term frequency scaled by a logarithmic factor, while the entropy of the term within the document collection is used for the global weight.

Step 5: Application of an algebraic model. After having built a TDM, different algebraic models can be used to process it. The simplest one is to use the TDM as is, *i.e.,* to use a traditional Vector Space Model (VSM). VSM is the simplest IR-based technique applied to traceability recovery. In VSM, artifacts are represented as vectors of terms (*i.e.,* columns of the TDM) that occur within artifacts in a repository [3]. VSM does not take into account relations between terms of the artifacts vocabulary. Alternatively, one can use Latent Semantic Indexing (LSI) [12], which is an extension of the VSM. It was developed to overcome the synonymy and polysemy problems, which occur in VSM model. LSI explicitly takes into account the dependencies between terms and between artifacts, in addition to the associations between terms and artifacts. To exploit information about co-occurrences of terms, LSI applies Singular Value Decomposition (SVD) [8] to project the original TDM into a reduced space of concepts, and thus limit the noise due to a high number of original dimensions (unique terms). A crucial factor that determines the performances of LSI is the choice of the number of concepts, *i.e.,* the k parameter. Then, there are other methods that treat documents as probability distributions of terms, among others JS [7] and Latent Dirichlet Allocation (LDA) [4]. The latter also requires

a calibration of different parameters, such as number of topics and the Dirichlet distribution parameters.

Step 6: Use of a distance (or similarity) measures. The last step of the IR process aims at comparing documents, *e.g.,* requirements and source code in traceability link recovery, queries and source code in feature location, bug report pairs in duplicate bug report detection. This can be done using different similarity measures. For example, one can rely on the cosine similarity, the Jaccard similarity, or the Dice (symmetric or asymmetric similarity) coefficient.

B. Traceability Link Recovery

For IR-based traceability link recovery tasks [1] [33], the typical artifacts used for generating the corpus consist of documentation artifacts, such as requirements, use cases or class diagrams and source code elements, such as classes or methods. These artifacts depend on the traceability recovery task. For example, to recover traceability links between use cases and source code classes, the use cases are used as queries (or *source artifacts*) and the classes are used as *target artifacts*. All these artifacts are typically preprocessed by (i) removing special characters, (ii) splitting identifiers, (iii) removing stop words that are common in language as well as words that appear frequently in the templates of the source artifacts (*e.g.,* "use case number", "actor", etc.) and (iv) stemming. The most used weighting schema for the TDM is the standard *tf-idf*, while VSM and LSI are among the most used IR techniques. The cosine similarities between all the source artifacts and all the target artifacts are used to rank the potential candidate links, which are presented to the developer for investigation to decide if they are correct or not.

C. Identification of Duplicate Bug Reports

For the task of detecting duplicate bug reports [48] [50], the primary source of information for constructing the corpus consists of the information extracted from issue tracking systems. Each document of the corpus (*i.e.,* each bug) typically consists of the title (*e.g.,* short description of the issue), the description, and in some cases, attributes such as the project name, component name, severity, priority, etc. In these documents, different weights could be assigned to the previously enumerated elements (*e.g.,* a title could be weighted more than a description). The *source artifacts* are new, unassigned bugs for which the developer is trying to find similar bugs, and the *target artifacts* are existing bugs, which were assigned to developers or resolved. Similarly to the other tasks, the corpus is preprocessed using the standard steps (*e.g.,* removing any sentence punctuation marks, splitting identifiers, removing stop words and stemming). The similarity measure between the source (*i.e.,* new) bugs and the target (*i.e.,* existing) bugs, which is computed using an IR technique (*e.g.,* VSM), is used to rank the list of bugs presented to the developer for manual inspection.

III. The Proposed Approach: GA-IR

This section describes how we use GAs to instantiate an IR process to solve software engineering problems (GA-IR). First, we treat the instantiation of the process described in Section II-A as a search-based optimization problem and then describe the GA approach. Subsequently, we illustrate how we evaluate the fitness function during the GA evolution.

Genetic Algorithms (GAs) [24] are a stochastic search technique inspired by the mechanism of a natural selection and natural evolution. A GA search starts with a random population of solutions, where each individual (*i.e., chromosome*) of a population represents a solution of the optimization problem. The population is evolved toward better solutions through subsequent generations and, during each generation, the individuals are evaluated based on the *fitness* function that has to be optimized. For creating the next generation, new individuals (*i.e., offsprings*) are generated by (i) applying a *selection operator*, which is based on the fitness function of the individuals to be reproduced, (ii) recombining, with a given probability, two individuals from the current generation using the *crossover operator*, and (iii) modifying, with a given probability, individuals using the *mutation operator*.

In this paper we propose to use a GA to automatically assemble and instantiate parameterized IR-based solutions for SE tasks. Specifically, we use GA to (i) instantiate the IR technique represented in the chromosome of an individual, (ii) execute the IR technique, *i.e.,* processing the documents using that IR process, and (iii) compute the GA fitness function by clustering the processed documents and evaluating the internal clustering quality.

A. Use GA to instantiate IR processes

The first choice in the design of a GA for an optimization problem is the representation of candidate solutions that must be encoded in a suitable form, known as *chromosome representation*. As explained in Section II, an IR process (see Fig. 1) consists of a sequence of given steps or components, thus it naturally lends itself to be represented as a chromosome. In GA-IR, the chromosome (see Fig. 2) is a vector, where each cell (*gene*) represents a phase of the IR process, and can assume as a possible value any of the techniques/approaches available for that phase. For example, for the morphological analysis we could have "no stemming", "Porter", or "Snowball". Note that, since some steps require multiple decisions, they are represented as two genes. In particular, the term extraction has two genes, one related to what kind of characters to prune, another related to how to split compound identifiers. In principle, further, more complex configurations can be foreseen. The right-hand side of the chromosome encodes the parameter settings for the used IR methods. In this paper we consider two widely used IR-methods, namely VSM and LSI, which already prove to be very successful for the investigated software engineering tasks [20], [41], [16], [42], [26], [47], [43], [40], [44]. In terms of calibration, LSI requires to set the number of concepts (k) to which the term-by-document space will be reduced, while VSM has no additional parameters.

GA-IR is based on a simple GA with elitism of two individuals, *i.e.,* the two best individuals are kept alive across generations. The GA initial population is generated by randomly choosing the value of each gene of each individual. The *selection* of the individuals to be reproduced is performed using the *Roulette Wheel* selection operator, which elects individuals to reproduce pseudo-randomly, giving higher chances to individuals with higher fitness. The *crossover* operator is the

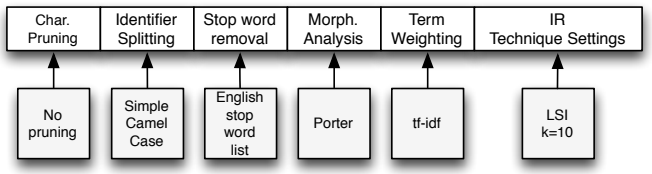


Fig. 2. GA-IR chromosome representation.

single-point crossover, which, given two individuals (parents) p_1 and p_2, randomly selects a position in the chromosome, and then creates two new individuals (the offspring) o_1 composed of the left-side of p_1, and the right-side of p_2, and o_2 composed of the left-side of p_2 and the right-side of p_1. The *mutation* operator is the uniform mutation, which randomly changes one of the genes with one of the admissible values. The GA terminates after a fixed number of generations or when the fitness function cannot be improved further (*i.e.,* GA converged).

B. Measuring the quality of assembled IR approach instances

Another important step in the design of a GA is the definition of the fitness function. In GA-IR we need to define a fitness function able to estimate the performances of the resulting IR technique. Thus, the fitness function evaluation needs to be unsupervised (*i.e.,* it does not require a labeled training set or an oracle), in order to make it task-independent.

When applying IR methods, such as LSI or VSM, to extract textual information from software artifacts, such techniques implicitly cluster the software documents on the basis of their textual similarities. Such different clusterings can be obtained by using various numbers of latent concepts used for modeling the concept space, independently from the used pre-processing techniques. Specifically, given that applying LSI on the term-document matrix results in the following decomposition (through Singular Value Decomposition):

$$TM = U \cdot \Sigma \cdot V^T$$

where U is the documents-to-concepts eigenvector matrix, Σ the eigenvalue matrix, and V^T the transposed terms-to-concepts eigenvector matrix. Once the LSI space is reduced to k concepts (k is indeed the LSI configuration parameter) the decomposition becomes:

$$TM_k = U_k \cdot \Sigma_k \cdot V_k^T$$

Given the matrix U_k, we assign each document i to its dominating concept j, *i.e.,* the concept with the highest $U_k(i,j)$ value. For the traditional VSM, we apply a similar approach, however, without reducing the space, but considering k equal to the rank of the term-document matrix.

We conjecture that there is a strong relationship between the performances obtained by an IR technique on software corpora and the quality of the internal clusters produced. Indeed, if the quality of the clusters produced by an IR-process is poor, it means that the IR process was not able to correctly extract the most important concepts from the software corpus and the documents, which are more similar to each other, are assigned to different clusters. Similarly to what has been done in previous work when calibrating LDA [38], we use the Silhouette coefficient [28] to measure the quality of clusters, since it provides only one scalar value combining *cohesion* and *separation* of clusters. In particular, the Silhouette coefficient is computed for each document using the concept of centroids of clusters. Let C be a cluster; its centroid is equal to the mean vector of all the documents belonging to C, *i.e.,* $Centroid(C) = \sum_{d_i \in C} d_i / |C|$. Starting from the definition of centroids, the Silhouette coefficient is computed for each document d_i as the following:

$$s(d_i) = \frac{b(d_i) - a(d_i)}{\max\left(a(d_i), b(d_i)\right)}$$

where $a(d_i)$ is the separation (measured as the maximum distance from d_i to the centroid of its cluster) and $b(d_i)$ is the cohesion (represented as the minimum distance from d_i to the centroids of the clusters not containing d_i). The value of the Silhouette coefficient ranges between -1 and 1.

A good cluster has a positive Silhouette coefficient because it corresponds to the case in which $b(d_i) > a(d_i)$, *i.e.,* the mean distance to other documents in the same cluster is less than the minimum distance to other documents in other clusters. We used the cosine of the angle between vectors for measuring the distance between documents, since in LSI the documents are represented as vectors in the concepts space. In the end, the overall measure of the quality of clustering $C = \{C_1, \ldots, C_k\}$, that is our fitness function, is computed by the mean Silhouette coefficient of all the documents.

IV. Empirical Evaluation Design

The *goal* of our study is to investigate whether GA-IR is able to instantiate IR techniques that are able to effectively solve SE tasks, while the *quality focus* is represented by the performances of the IR-based processes in terms of accuracy and completeness. The *perspective* is of researchers interested in developing an automatic approach to assemble IR processes for solving specific SE tasks. The *context* of the study consists of (i) *two SE tasks*, namely traceability link recovery, and identification of duplicate bug reports, and (ii) their corresponding *objects* (*i.e.,* datasets on which the tasks are applied). Specifically, the study aims at answering the following research questions (RQs) that have been addressed in the context of the two SE tasks considered in our study:

RQ1: *How do the IR techniques and configurations instantiated by GA-IR compare to those previously used in literature for the same tasks?* This RQ aims at justifying the need for an automatic approach that calibrates IR approaches for SE tasks. Specifically, we analyzed to what extent the techniques instantiated by GA-IR for solving a specific task are able to provide better performances as compared to those configured with "ad-hoc" settings. Our conjecture is that, with proper settings, the performances could be sensibly improved because, in many cases, the IR-based techniques have been severely under-utilized in the past.

RQ2: *How do the settings of IR-based techniques instantiated by GA-IR compare to an ideal configuration?* We empirically identified the configuration that provided the best

results as compared to a specific oracle. For instance, in the case of traceability link recovery, we identified the configuration that provided the best performances in terms of correct and incorrect links recovered. Clearly, one can build such a configuration only with available labeled data set, by using a combinatorial search among different treatments and by evaluating each combination against the oracle in terms of precision and recall.

We call such a configuration "ideal", because it is not possible to build a priori (*i.e.,* without the availability of a labeled training set) a configuration providing better performances than that. The performances achieved by the techniques instantiated by GA-IR are then compared to those achieved with the ideal configuration, to investigate how far off is the GA-IR configuration from the best possible performances that one can achieve.

A. GA-IR Implementation and Settings

GA-IR has been implemented in *R* [45] using the `GA` library. Every time an individual needs to be evaluated, we process documents using features available in the `lsa` package which allows applying all the pre-processing steps, while for computing the SVD decomposition we used the fast procedure provided by the `irlba` package for large and sparse matrices. As for the GA settings, we used a crossover probability of 0.8, a uniform mutation with probability of $1/n$, where n is the chromosome size ($n = 7$ in our case). We set the population size equal to 50 individuals with elitism of two individuals. As a stop condition for GA, we terminate the evolution if the best fitness function value does not improve for ten consecutive generations or when reaching the maximum number of generations equals to 100 (which, however, was never reached in our experiments). All these settings are commonly used in the genetic algorithms community. In addition, to address the intrinsic randomness of GAs, for each task and for each dataset, we performed 30 independent runs, storing the best configuration and the relative best fitness function value—*i.e.,* the Silhouette coefficient—for each run. Finally, among the obtained configurations, we consider the one that achieves the median fitness function—across the 30 independent runs—for the best individual in the last generation. Finally, Table I summarizes the possible values for each gene of the chromosome used in our experimentation. Clearly, the number of possible values can easily be extended (*e.g.,* different stemming, different weighting schemas).

B. Task 1: Traceability Link Recovery

For this task, we use the IR methods to recover traceability links between high level artifacts (*e.g.,* use cases) and source code classes. The experiment has been conducted on the software repositories of three projects, EasyClinic, eTour, and iTrust. EasyClinic is a system used to manage a doctor's office, while eTour is an electronic touristic guide. The documentation, source code identifiers, and comments for both systems are written in Italian. Both EasyClinic and eTour have been developed by final year Master students at the University of Salerno (Italy). iTrust is a medical application used as a class project for Software Engineering courses at North Carolina State University[1]. All artifacts consisting of use cases and

[1]http://agile.csc.ncsu.edu/iTrust/wiki/doku.php?id=tracing

TABLE I. VALUES OF THE GENES (STEPS OF THE IR-BASED TECHNIQUE) FOR GA-IR.

STEP	IMPLEMENTATIONS
Character pruning	keep special characters and digits remove special characters, keep digits remove both special characters and digits
Identifier splitting	do not split simple Camel Case split Camel Case keep compound split
Stop word removal	do not remove stop words English/Italian stop word removal English/Italian stop word removal + remove $\leq$ 2 char words English/Italian stop word removal + remove $\leq$ 3 char words
Morphological analysis	no stemming Porter stemming Snowball stemming
Term weighting	Boolean *tf* *tf-idf* $log_{ij} = \log\left(tf_{ij} + 1\right)$ *tf*-entropy
Algebraic model	VSM LSI
LSI k	$10 \leq k \leq rank(TDM)$

TABLE II. TASK 1 (TRACEABILITY LINK RECOVERY): CHARACTERISTICS OF THE DATASETS USED IN THE EXPERIMENT.

System	KLOC	Source Artifacts (#)	Target Artifacts (#)	Correct links
EasyClinic	20	UC (30)	CC (47)	93
eTour	45	UC (58)	CC (174)	366
iTrust	10	UC(33)	JSP (47)	58
UC: Use case, CC: Code class; JSP: Java Server Page				

Java Server Pages are written in English. Table II summarizes the characteristics of the considered software systems: the number and type of source and target artifacts, the source code size in KLOC, and the number of correct links between the source and target artifacts. These correct links are derived from the traceability matrix built and validated by the original developers. We consider such a matrix as the oracle to evaluate the accuracy of different traceability recovery processes.

To answer **RQ1**, we compared the accuracy of recovering traceability links achieved by the IR techniques assembled by GA-IR with the accuracy achieved by LSI and VSM on the same systems in the previously published studies where an "ad-hoc" corpus pre-processing and LSI configuration were used [10], [11]. We also compared the accuracy of recovering traceability links using different combinations of pre-processing steps ($3 \times 3 \times 4 \times 3 \times 5$=540, see Table I) for both VSM and LSI. For LSI, we also consider different number of concepts by varying k from 10 to a maximum number of topics, which is 77 for EasyClinic, 176 for eTour and 80 for iTrust. We also exercised all possible combinations of pre-processing steps with such values. Thus, the total number of trials performed on EasyClinic, eTour, and iTrust were 36,720, *i.e.,* $540 \cdot$ (77-10+1), 89,640 and 37,800, respectively. With such an analysis we are able to identify the configuration which provides the best recovery accuracy (as compared to our oracle) between all the possible configurations aiming at estimating the ideal configuration of the IR-based traceability recovery process. Then, in order to answer **RQ2**, we compared the performances achieved with such an ideal configuration with the performances achieved with the configuration identified by GA-IR.

For both **RQ1** and **RQ2**, the performances of the GA-IR approach, the baseline approach, and the ideal approach are evaluated and compared by using two well-known metrics in

the IR field, namely precision and recall [3]. The precision values achieved for different levels of recall (for each correct link) by the different IR processes are then pairwise-compared using the Wilcoxon rank sum test. Since this requires performing three tests for each system, we adjusted the p-values using Holm's correction procedure [25]. Finally, we use the average precision metric [3] for comparing the performances of the different IR processes. We used the average precision since it provides a single value that is proportional to the area under precision-recall curve achieved for each ranked list of candidate links. This value is the mean of the precision over all the correct links. Hence, it combines both precision and recall into a single performance scalar value.

C. Task 2: Duplicate Bug Report Identification

For this task, we used VSM and LSI to identify duplicate bug reports from a set of existing reports. In essence, we used two selected IR methods for computing the textual similarity between new bug reports and existing bug reports using their descriptions. The textual corpora is the one composed of (i) the title of the bug, and (ii) the double weighted title and the description of the bug.

Besides the textual similarity, as suggested by Wang *et al.* [50], for each bug report in the analyzed corpus, we also generated an execution trace by following the steps to reproduce the bug (such steps are available in the bug description). Using this information we built a bug-to-method matrix, where each bug represents a column, and each method represents a row. The matrix has binary values, *i.e.,* a generic entry (i, j) is one, if the i^{th} method in the corresponding row appears in the execution trace of the j^{th} bug; zero otherwise. This information can be used to identify bugs having similar execution traces. Such information can complement textual similarity in order to identify duplicate bug reports, *i.e.,* bugs having similar execution traces are candidate to be the same bug. We then apply VSM and LSI also on the bug-to-method matrix and we compute the similarity between each bug (in terms of execution trace) using the *Execution-information-based Similarity*. The final similarity between each pair of bug reports is given by averaging the textual similarity and the similarity of the execution traces of the two bugs.

The design of our study is based on the study introduced by Wang *et al.* [50], however, it is different in several important aspects, such as the IR techniques being used (we used both VSM and LSI, while they used VSM only), dataset, type of execution traces and method signatures. For example, Wang *et al.* used 220 bug reports of Eclipse 3.0 posted on June 2004 and 44 duplicate pairs as well full execution traces with method signatures. For our evaluation, we used 225 bugs (of the same system and posted in the same period) with 44 duplicate pairs, and marked execution traces without method signatures. For collecting the data, even though we followed the methodology described in their approach, we do not have information about the exact set of bugs used by Wang *et al.* [50]. Moreover, the execution traces we collected are most likely different since this was a manual collection process. In addition, our JPDA[2] instrumentation did not record the method signatures for the executed methods. In summary, we created a realistic reference approach for Task 2, however this may not fully correspond to the one in Wang *et al.* [50].

For each duplicate pair, we compute the similarity between the oldest submitted bug (among those two) and the remaining 224 bug reports in the corpus. We compute the accuracy of detecting all the pairs using the Recall Rate (RR) [48], [50]. Then, to address **RQ1**, we compare the RR of the configuration produced by GA-IR against the RR of a "baseline" configuration produced by using the preprocessing of Wang *et al.*, and by applying VSM and LSI with an "ad-hoc" number of concepts used in traceability link recovery, *i.e.,* $k = 50\%$ of total number of documents [11]. For **RQ2**, we compare the RR generated by the configuration suggested by GA-IR against the RR of the best configuration produced by performing a combinatorial search on the preprocessing steps and k values.

Similarly to the other two tasks, the RR values at different cut points (*i.e.,* suggested list sizes) are pairwise-compared using the Wilcoxon rank sum test, adjusting the corresponding p-values using Holm's correction procedure.

V. Empirical Evaluation Results

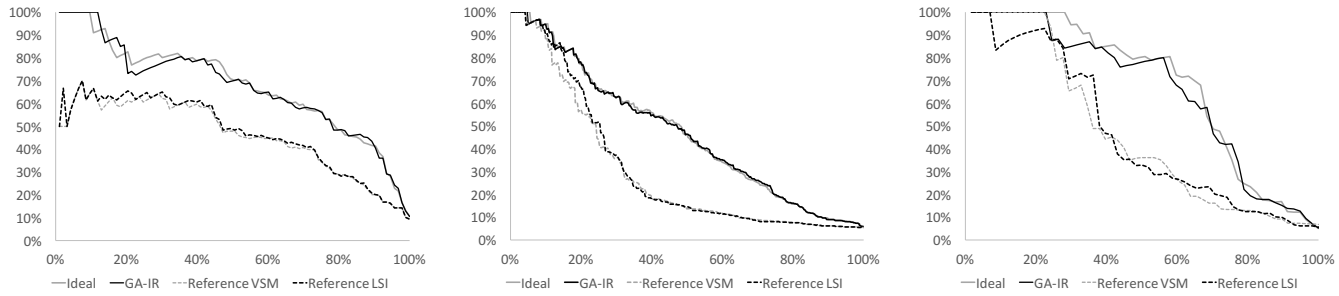
This section describes the results of our experiments conducted in order to answer the research questions stated in Section IV. We report the results for two different tasks in Section V-A and Section V-B respectively.

A. Task 1: Traceability Link Recovery

Fig. 3 reports the precision/recall graphs obtained using (i) the "ideal" IR configuration; (ii) the IR configuration identified by GA-IR; and (iii) an "ad-hoc" configuration (referred to as a reference) used in a previous study, *i.e.,* VSM and LSI used on the same dataset and for the same traceability recovery task. For all three systems, namely EasyClinic, eTour and iTrust, GA-IR was able to obtain a precision and recall rate close to the one obtained by the combinatorial (*i.e.,* "ideal") configuration. It is important to emphasize that the configuration identified by GA-IR used no information about the oracle for computing the result, whereas the combinatorial search used more than 30K different configurations for all the three projects to identify the best configuration, and its performance was evaluated based on the oracle. In other words, among those $> 30K$ configurations, we chose for comparison the one that produced the best results based on the oracle.

Based on Fig. 3, when comparing the performance achieved by GA-IR with those of the reference configurations, we can observe a significant improvement in all the cases. These results are also confirmed by the average precision obtained by the three different treatments (see Table III). Indeed, the average precision obtained by GA-IR is very close to the "ideal" (combinatorial) one. For example, for EasyClinic the average precision obtained with GA-IR is 66.92%, which is slightly lower than the average precision obtained by the "ideal" search, which is 67.47%. The same can be observed for eTour and iTrust. Moreover, the difference in terms of average precision with respect to the combinatorial configuration is lower than 1%. However, the improvement obtained with respect to the reference configurations is of about 20% in terms of average precision. For example, for EasyClinic the average precision obtained by GA-IR is 66.92%, whereas the average

[2]http://docs.oracle.com/javase/7/docs/technotes/guides/jpda/jpda.html



(a) EasyClinic (b) eTour (c) iTrust

Fig. 3. Traceability link recovery: precision/recall graphs.

TABLE III. COMPARISON OF TRACEABILITY LINK RECOVERY PERFORMANCES: AVERAGE PRECISION VALUES.

System	**Ideal**	**GA-IR**	**Ref. LSI** [10], [11]	**Ref. VSM** [10], [11]
EasyClinic	67.47	66.92	46.78	45.50
eTour	47.01	46.94	30.93	29.94
iTrust	68.84	68.13	45.47	46.07

TABLE IV. COMPARISON OF TRACEABILITY LINK RECOVERY PERFORMANCES (PRECISION): RESULTS OF WILCOXON RANK SUM TEST.

	EasyClinic	**eTour**	**iTrust**
GA-IR $>$ Ideal	0.99	1	0.99
GA-IR $>$ VSM-Reference [10], [11]	$<$ **0.01**	$<$ **0.01**	$<$**0.01**
GA-IR $>$ LSI-Reference [10], [11]	$<$ **0.01**	$<$ **0.01**	$<$**0.01**
Ideal $>$ VSM-Reference [10], [11]	$<$ **0.01**	$<$ **0.01**	$<$ **0.01**
Ideal $>$ LSI-Reference [10], [11]	$<$ **0.01**	$<$ **0.01**	$<$ **0.01**

precision values obtained by the reference configurations (*i.e.,* the ones that used an "ad-hoc" configuration) are 46.78% and 45.50% for LSI and VSM respectively. The findings presented in Fig. 3 and Table III are also confirmed by statistical tests (see Table IV), which show that for all three systems, there is no statistical difference between the results produced by GA-IR and the "ideal" search, but there is a statistical difference between the results produced by GA-IR and the references (baselines). In summary, GA-IR outperforms the baselines and the differences are statistically significant.

B. Task 2: Duplicate Bug Report Identification

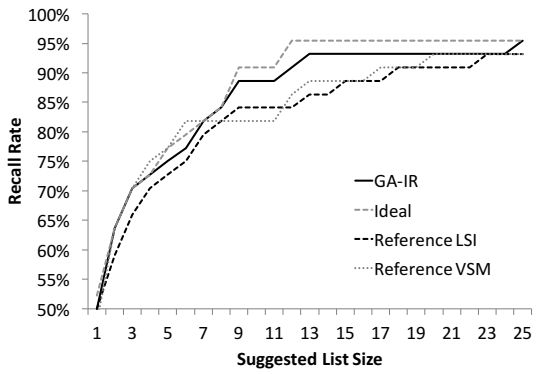
For the identification of duplicate bug reports, we used two different corpora, referred as *Short* and *2ShortLong* (as suggested by [50]). In the former case, each bug is characterized by only the bug title, while in the latter case we used both the title and the bug description where the title is weighted twice as much as the bug description. Note that in both cases we combined textual information with information belonging to execution traces.

Fig. 4 reports the recall rate for the results produced by using four different configurations, *i.e.,* GA-IR, "ideal" and two reference configurations. The reference configurations are obtained by applying a standard corpus preprocessing that is typical to bug duplicates [50] and other SE tasks [30] and using two IR methods, *i.e.,* LSI and VSM. For LSI, we also set the number of concepts (k) equal to half the number of documents for LSI [11]. When the corpus consists of both titles and bug descriptions, (*i.e., 2ShortLong*), GA-IR achieved approximately the same RR values as the ideal with only few exceptions to this rule, *i.e.,* for cut points equal to 4,10, and 11. However, in these few cases, the differences are negligible because these are always lower than 2%. From Fig. 4-(b) we can also notice that GA-IR produces better results in terms of RR as compared to the reference baselines. In particular, the RR for GA-IR is higher than the RR of the LSI and VSM with "ad-hoc" configurations, with an improvement raging between 2% and 16%.

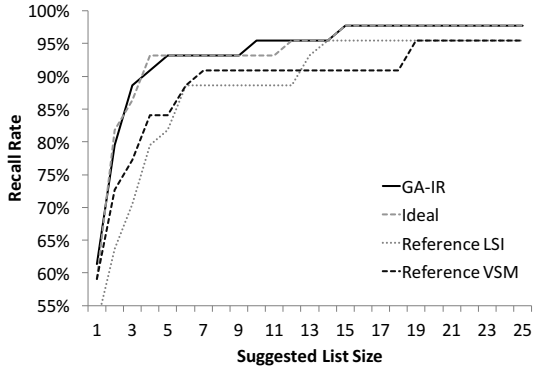
When the corpus consists of bug report titles only (*i.e., Short*) we can observe that there is no difference between RR scores achieved by GA-IR and the "ideal" configuration (based on the oracle) for cut points lower than 10. For larger cut-points, the "ideal" configuration yields higher recall rate than GA-IR with a difference of 2%. Only for cut point equal to 25 (*i.e.,* the last one) the two configurations report again the same RR scores. However, when comparing GA-IR with the two reference baselines (*i.e.,* LSI and VSM with "ad-hoc" configurations) we can notice that GA-IR recall rate is either equal or higher than the reference rate. Indeed, for cut points ranging from 8 to 19, the GA-IR recall rate is approximately 2-7% higher than recall rate achieved by both LSI and VSM with "ad-hoc" configuration. While for cut-points lower than 8 or greater than 20, GA-IR's and reference's recall rates are the same as observed from the graph in Fig. 4-(b).

Comparing the results reported in Fig. 4-(a) and Fig. 4-(b), we can observe that the performance of all the configurations ("ideal", GA-IR and reference configurations) are higher when the corpus is represented by both bug title and description, *i.e.,* for the *2ShortLong* corpus. We also observe that the gap between GA-IR and the reference is much higher for the *2ShortLong* corpus as opposed to the *Short* corpus, and one explanation for this finding could be the fact that, on one hand, when additional information is added to the corpus (*i.e.,* a long description), the benefits of properly calibrating an IR technique (*i.e.,* using GA-IR) are much more effective. On the other hand, when the corpus is particularly limited, as in the case of bug titles that contains only few words, a proper calibration of IR techniques can result in smaller benefits.

Table V reports the results of the Wilcoxon test (adjusted p-values) for all combinations of the techniques (statistically significant results are highlighted in bold face). The results indicate that on both *Short* and *2ShortLong* corpora GA-IR statistically outperforms the two reference baselines, *i.e.,* VSM and LSI instantiated with "ad-hoc" configuration. Moreover, there is no significant difference between GA-IR and the "ideal" configuration for the *Short* corpus. However, for the



(a) Short Corpus



(b) 2ShortLong Corpus

Fig. 4. Recall Rate graphs for Eclipse, with suggested list size raging between 1 and 25.

TABLE V. COMPARISON OF BUG REPORT DUPLICATION: RESULTS OF THE WILCOXON RANK SUM TEST.

	Short Corpus	2ShortLong Corpus
GA-IR > Ideal	< **0.01**	0.48
GA-IR > LSI-Reference	< **0.01**	< **0.01**
GA-IR > VSM-Reference	< **0.01**	< **0.01**
Ideal > VSM-Reference	< **0.01**	< **0.01**
Ideal > VSM-Reference	< **0.01**	< **0.01**

2ShortLong corpus the "ideal" configuration performs significantly better than GA-IR.

C. Detailed description of the experimented IR processes

Table VI details the specific IR-configurations (*i.e.,* the preprocessing steps, IR technique and IR-technique parameter values) that were identified by GA-IR, the "ideal" search and the references. The rows are grouped by task (*i.e.,* traceability link recovery, and identification of duplicate bug reports), dataset (or systems) and approach (*i.e.,* GA-IR, "ideal", and references), and the columns represent the steps of the IR-process, or the gene type illustrated in Table I.

From the fourth column of Table VI we notice that for both tasks, and for all datasets and approaches, LSI was the IR-technique that was chosen by GA-IR and the "ideal" search, which tested LSI, and VSM. For most baselines, LSI was the suggested technique and was configured with the dimensionality reduction factor of 50% the number of documents from the corpus for both traceability link recovery and identification of duplicate bug reports. It is important to mention that there were other IR-configurations that used VSM and produced results close to the ones presented in previous sub-section, however, in the end LSI-based configurations outperformed them. Regarding the number of topics (k) selected for LSI, for all three approaches we observe a clear pattern: (i) the reference configuration uses a very different value than the "ideal" or GA-IR; (ii) the k values selected by GA-IR and "ideal" are very close to each other. This would indicate that GA-IR allows us to instantiate an IR process that is close to the one identified by the "ideal" one.

When comparing the preprocessing steps for character pruning (see Table VI columns five and six), we observe that in most of the cases the special characters were removed, and in some cases, the "ideal" and GA-IR approaches include the digits in the corpus. The reference configuration always removed the digits, because the assumption was that their contribution to providing meaning to the IR model that processes a corpus from source code was limited. However, our findings show that this assumption is not always true: for both the traceability link recovery and identification of duplicate bug report tasks (which include in their corpora use cases and bug descriptions respectively) the "ideal" and GA-IR approaches often choose the option to include the digits, as they might carry some meaning, which in turn would help improve the results over the baseline (which always removed the digits). For the choice of splitting identifiers, in the majority of cases (*i.e.,* 16 out of 20), the standard Camel Case splitting algorithm was applied (see Table VI column seven). Only two exceptions to the rules are obtained for iTrust with the "ideal" IR process and for Eclipse (*Short* corpus) with both the "ideal" IR process and GA-IR where the identifiers were split using Camel Case, as well as keeping the original (compound) identifiers.

If we compare the preprocessing steps of removing stop words and stemming (see Table VI columns seven and eight) we observe that in all the cases a standard list of stop words was used (*i.e.,* no approach was configured by keeping programming language keywords or identifiers frequently used in English, such as articles, prepositions). In addition, in some cases, GA-IR and the "ideal" search choose to also remove identifiers with less than two or three characters, in order to reduce the noise from the IR model. In terms of stemmers, every approach used either the Snowball or Porter stemmer (*i.e.,* no approach was configured with no stemming at all), with the only exception of the two references (*i.e.,* VSM and LSI with "ad-hoc" configuration) for identification of duplicate bug report tasks where no stemmer is used [50].

For all approaches, the chosen function to compute the document to document similarity was the cosine similarity (see last column in Table VI). Moreover, for the majority of cases (*i.e.,* 19 out of 20) the tf-idf measure was chosen as the preferred weighting schema (see Table VI column 11). However, GA-IR choose the log weighting schema for iTrust. Although we can identify some patterns when analyzing each preprocessing step individually, we cannot determine a clear pattern when we take all the preprocessing steps and the configuration of the IR techniques as a whole, but we can observe a clear variation in the choices. The fact that not all the tasks and/or datasets require the same preprocessing

TABLE VI. COMPARISON OF DIFFERENT IR PROCESSES PROVIDED BY GA-IR, COMBINATORIAL AND REFERENCE. TABLE ABBREVIATIONS: CC = CAMEL CASE; CC & KC = CAMEL CASE PLUS KEEP-COMPOUND IDENTIFIER.

Task	System	Strategy	IR Method	Special Character	Digit Chars	Term Splitting	Stopword List	Stopword Function	Stemming Stemming	Weight Schema	Similarity Function
Task 1	EasyClinic	Ideal	LSI ($k = 60$)	Remove	Include	CC	Yes	No	Snowball	tf-idf	Cos
		GA-IR	LSI ($k = 53$)	Remove	Remove	CC	Yes	$\leq$ 2 chars	Snowball	tf-idf	Cos
		Reference	LSI ($k = 37$)	Remove	Remove	CC	Yes	$\leq$ 3 chars	Snowball	tf-idf	Cos
		Reference	VSM	Remove	Remove	CC	Yes	$\leq$ 3 chars	Snowball	tf-idf	Cos
	eTour	Ideal	LSI ($k = 170$)	Remove	Include	CC	Yes	No	Snowball	tf-idf	Cos
		GA-IR	LSI ($k = 149$)	Remove	Remove	CC	Yes	No	Porter	tf-idf	Cos
		Reference	LSI ($k = 87$)	Remove	Remove	CC	Yes	No	Snowball	tf-idf	Cos
		Reference	VSM	Remove	Remove	CC	Yes	No	Snowball	tf-idf	Cos
	iTrust	Ideal	LSI ($k = 75$)	Remove	Include	CC & KC	Yes	$\leq$ 2 chars	Snowball	tf-idf	Cos
		GA-IR	LSI ($k = 79$)	Remove	Include	CC	Yes	$\leq$ 3 chars	Porter	log	Cos
		Reference	LSI ($k = 40$)	Remove	Remove	CC	Yes	No	Snowball	tf-idf	Cos
		Reference	VSM	Remove	Remove	CC	Yes	No	Snowball	tf-idf	Cos
Task 2	Eclipse Short	Ideal	LSI ($k = 124$)	Include	Include	CC & KC	Yes	No	Porter	tf-idf	Cos
		GA-IR	LSI ($k = 119$)	Include	Include	CC & KC	Yes	No	Porter	tf-idf	Cos
		Reference	LSI ($k = 112$)	Remove	Include	No	Yes	No	Porter	tf-idf	Cos
		Reference	VSM	Remove	Include	No	Yes	No	Porter	tf-idf	Cos
	Eclipse 2ShortLong	Ideal	LSI ($k = 180$)	Remove	Include	CC	Yes	No	Porter	tf-idf	Cos
		GA-IR	LSI ($k = 185$)	Remove	Include	CC	Yes	$\leq$ 3 chars	Porter	tf-idf	Cos
		Reference	LSI ($k = 112$)	Remove	Remove	No	Yes	No	Porter	tf-idf	Cos
		Reference	VSM	Remove	Remove	No	Yes	No	Porter	tf-idf	Cos

steps confirms the findings of Falessi *et al.* [18] that there is no unique IR configuration that can be efficiently applied to all the tasks and all the datasets. Thus, GA-IR plays a key role in considering all the potential configurations of an IR-technique, and all their potential interactions as a whole (as opposed to considering them individually) and recommends the configuration that is most suited for a dataset, as each dataset is unique. We observed that besides recommending the preprocessing steps, GA-IR was able to recommend the number of topics k, and the overall IR configuration was able to produce better results than the baseline. Moreover, our comparison between GA-IR and the "ideal" search indicates that GA-IR was able to find a suitable configuration (using the Silhouette coefficient of the underlying model) that is close to the configuration identified by the "ideal" search, which used the oracle to identify the best configuration.

VI. THREATS TO VALIDITY

Threats to *construct validity* concern the relationship between theory and observation. For the two tasks investigated, we evaluated the performances of GA-IR and of alternative approaches using well-established metrics, namely precision and recall, effectiveness measure and Recall Rate, as well as oracles already used and validated in previous studies [20], [50]. We used as a baseline for comparison the performances achieved by IR techniques and settings used in previous papers. As for detecting duplicate bug reports, it was not possible to fully replicate the approach by Wang *et al.* [50] due to unavailability of all the required information. However, Section IV explains the details and the rationale for using a baseline for comparison for such a task.

Threats to *internal validity* are related to co-factors that could have influenced our results. We limited the influence of GA randomness by performing 30 GA runs, and considering the configuration achieving the median performance.

Threats to *conclusion validity* concern the relationship between a treatment and an outcome. To support our claims, we used a non-parametric statistical test, *i.e.,*, Wilcoxon rank sum test. Since multiple tests were performed on the same software projects, we also use the Holm's correction to adjust p-values.

Threats to *external validity* concern the generalization of our results. First, as explained in Section IV, for a better control over the independent and dependent variables of the study we considered only two IR methods (*i.e.,* LSI and VSM), although as illustrated in Section III the approach can be easily extended to consider and search among alternative IR methods and their configurations. In addition, we consider only a subset of the possible treatments for the various phases such as term extraction, stop words removal, stemming, and term weighting. Although the chosen treatments are representative of the ones used in literature, it is worthwhile to investigate other possibilities. Finally, while we applied GA-IR to assemble and calibrate IR-based techniques on artifacts in two SE tasks, it would be worthwhile to scrutinize GA-IR in the context of other SE tasks.

VII. RELATED WORK

This section describes related work concerning the importance of choosing proper treatments for different phases of an IR process when applied to SE tasks. It also reports approaches aimed at suggesting calibrations for specific IR techniques.

Falessi *et al.* [18] empirically evaluated the performance of IR-based duplicate requirement identification on a set of over 983 requirement pairs coming from industrial projects. To this aim, they instantiated 242 IR processes, using various treatments for stemming, term weighting, IR algebraic method, and similarity measures. Their study shows that the performances of different IR configurations for detecting duplicate requirements vary significantly. Also, Thomas *et al.* [49] studied the impact of classifier configuration in a bug localization process. They considered 3,172 configurations, where each configuration on one hand includes different steps of the IR-based indexing, and on the other hand different kinds of machine learning classifiers. The work by Falessi *et al.* [18] and Thomas *et al.* [49] motivate our research, as it empirically shows that instantiating an appropriate IR process is crucial to achieving good performances. However, while they do not propose an approach for choosing the most suitable

technique/configuration, our GA-IR approach searches for a (near) optimal IR technique, which is also properly configured, using GAs, and above all, it is able to do it without the availability of an oracle.

Moreno *et al.* [36] investigated the effects of different text retrieval configurations on feature location approaches, and proposed an approach named QUEST to perform a query-based identification of a text retrieval configuration. In other words, their approach determines the text retrieval configuration most suitable for a given query. Differently from our case, their approach is focused on choosing among a limited number (21) of configurations (therefore, the choice is performed exhaustively), and to make the choice dependent on the query. Instead, GA-IR can choose among a very high (and virtually unlimited) number of configurations by relying on a search-based optimization technique.

In our previous work [38] we proposed LDA-GA, a GA-based approach to automatically calibrate the parameters of LDA in the context of three software engineering tasks, namely traceability link recovery, feature location and software artifact labeling. In contrast to LDA-GA, GA-IR assembles and configures the entire IR process, rather than just tuning the parameters of a specific IR algebraic method. More specifically, our results are in concordance with the findings of Falessi *et al.* [18], which illustrate that performances of IR processes depend on the choices made for the various phases of IR processing, rather than just on the appropriate choice and calibration of the IR algebraic model.

The literature also reports approaches for calibrating specific stages of an IR process, or parameters of specific algebraic IR techniques. Cordy and Grant have proposed heuristics for determining the "optimal" number of LDA topics for a source code corpus of methods, by taking into account the location of these methods in files or folders, as well as the conceptual similarity between methods [21]. Cummins [9] proposed to use genetic programming (GP) to automatically build term weighting formulae, using different combinations of *tf* and *idf*, that can be altered using functions such as logarithm. The similarity between our approach and Cummins' approach is the use of search-based optimization techniques to calibrate IR processes. Their approach was evaluated on a set of 35,000 textual documents, for a document search task. In contrast to our technique, their approach (i) focuses on term weighting only (whereas we focus on the whole process), and (ii) their approach is supervised, as the fitness function evaluation requires the availability of a training set (*e.g.,* labeled traceability links). Griffiths and Steyvers [22] propose a method for choosing the best number of topics for LDA among a set of predefined topics. Their approach consists of (i) choosing a set of topics, (ii) computing a posterior distribution over the assignments of words to topics, (iii) computing the harmonic mean of a set of values from the posterior distribution to estimate the likelihood of a word belonging to a topic, and (iv) choosing the topic with the maximum likelihood.

To the best of our knowledge, the most related work to ours is the approach by Lohar *et al.* [31], who proposed to use GA for customizing a process for traceability recovery. While we share the goal of configuring each step of the IR process, and the application to traceability link recovery, there are substantial differences between the two approaches:

1) While the approach of Lohar *et al.* [31] is supervised, *i.e.,* it configures the IR process using GA on a training set on which traceability links have been labeled, our approach is *unsupervised*, as it calibrates the GA based on the achieved clustering quality. As mentioned in the introduction, this can be valuable when a labeled training set is unavailable;
2) For this reason, our approach is task independent, as it is able to configure an IR process independently of the task on which the configured process has to be applied while the approach by Lohar *et al.* [31] is task dependent.

VIII. Conclusion and Future Work

The application of IR techniques to software engineering problems requires a careful construction of a process consisting of various phases, *i.e.,* term extraction, stop word removal, stemming, term weighting, and application of an algebraic IR method. Each of these phases can be implemented in various ways, and requires careful choice and settings, because the performances significantly depend on such choices [18].

This paper proposes the use of Genetic Algorithms to assemble and configure a (near) optimal IR process to be applied to given software artifacts, (*e.g.,* when processing artifacts to solve problems such as traceability link recovery or feature location). Noticeably, the proposed approach is *unsupervised* and *task independent*, as it evaluates the extent to which the artifacts can be clustered after being processed.

We applied the proposed approach—named as GA-IR—to two software engineering tasks, namely traceability link recovery and detection of duplicate bug reports. The results of our empirical evaluation indicate that:

- for traceability recovery, the IR processes assembled by GA-IR significantly outperform those assembled according to existing approaches from literature;
- for duplicate bug report detection, the obtained results do not always significantly improve the performances of the baseline approach, since such a baseline is already close to the ideal optimum;
- in most cases, the performances achieved by GA-IR are not significantly different from the performances of an ideal IR process that can be combinatorially built by considering all possible combinations of treatments for the various phases of the IR process, and by having a labeled training set available (*i.e.,* by using a supervised approach).

Our work in progress aims at extending the proposed approach in various ways. First, we plan to use a more sophisticated evolutionary algorithm, employing genetic programming (GP) to assemble different phases in different ways, including creating ad-hoc weighting schemata [9] or extracting ad-hoc elements from artifacts to be indexed, *e.g.,* only specific source code elements, only certain parts-of-speech. Last, but not least, we plan to apply GA-IR in the context of other software engineering tasks, and to make it available as a plugin to the *Tracelab* [27] environment, as done already for the automatic configuration of LDA [15].

REFERENCES

[1] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[2] G. Antoniol, Y.-G. Guéhéneuc, E. Merlo, and P. Tonella, "Mining the lexicon used by programmers during sofware evolution," in *Proc. of the 23rd IEEE International Conference on Software Maintenance*. Paris, France: IEEE Press, 2007, pp. 14–23.

[3] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

[4] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet Allocation," *The Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[5] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella, "On the role of the nouns in IR-based traceability recovery," in *Proc. of 17th IEEE International Conference on Program Comprehension*, Vancouver, British Columbia, Canada, 2009.

[6] B. Caprile and P. Tonella, "Restructuring program identifier names," in *Proc. of 16th IEEE International Conference on Software Maintenance*. San Jose, California, USA: IEEE CS Press, 2000, pp. 97–107.

[7] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Wiley-Interscience, 1991.

[8] J. K. Cullum and R. A. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*. Boston: Birkhauser, 1998, vol. 1, ch. Real rectangular matrices.

[9] R. Cummins, "The evolution and analysis of term-weighting schemes in information retrieval," Ph.D. dissertation, National University of Ireland, 2008.

[10] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Applying a smoothing filter to improve IR-based traceability recovery processes: An empirical investigation," *Information and Software Technology*, 2012.

[11] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artefact management systems using information retrieval methods," *ACM Trans. on Soft. Eng. and Methodology*, vol. 16, no. 4, 2007.

[12] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[13] F. Deissenboeck and M. Pizka, "Concise and consistent naming," *Software Quality Journal*, vol. 14, no. 3, pp. 261–282, 2006.

[14] B. Dit, L. Guerrouj, D. Poshyvanyk, and G. Antoniol, "Can better identifier splitting techniques help feature location?" in *The 19th IEEE International Conference on Program Comprehension, ICPC 2011, Kingston, ON, Canada, June 22-24, 2011*. IEEE Computer Society, 2011, pp. 11–20.

[15] B. Dit, A. Panichella, E. Moritz, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "Configuring topic models for software engineering tasks in tracelab," in *7th International Workshop on Traceability in Emerging Forms of Software Engineering, TEFSE 2013, 19 May, 2013, San Francisco, CA, USA*, 2013, pp. 105–109.

[16] B. Dit, M. Revelle, and D. Poshyvanyk, "Integrating information retrieval, execution and link analysis algorithms to improve feature location in software," *Empirical Software Engineering (EMSE)*, pp. 1–33, to. appear, 2012.

[17] S. T. Dumais, "Improving the retrieval of information from external sources," *Behavior Research Methods, Instruments and Computers*, vol. 23, pp. 229–236, 1991.

[18] D. Falessi, G. Cantone, and G. Canfora, "Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques," *IEEE Trans. Software Eng.*, vol. 39, no. 1, pp. 18–44, 2013.

[19] M. Gethers, B. Dit, H. Kagdi, and D. Poshyvanyk, "Integrated impact analysis for managing software changes," in *Proc. of the 34th IEEE/ACM International Conference on Software Engineering (ICSE'12)*, Zurich, Switzerland, June 2-9, 2012, pp. 430–440.

[20] M. Gethers, R. Oliveto, D. Poshyvanyk, and A.De Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in *Proc. of the 27th International Conference on Software Maintenance (ICSM'11)*. Williamsburg, VA, USA, Sept. 25-Oct. 1: IEEE Press, 2011, pp. 133–142.

[21] S. Grant and J. R. Cordy, "Estimating the optimal number of latent concepts in source code analysis," in *Proc. of the 10th International Working Conference on Source Code Analysis and Manipulation (SCAM'10)*, 2010, pp. 65–74.

[22] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proc. of the National Academy of Sciences*, vol. 101, no. Suppl. 1, pp. 5228–5235, 2004.

[23] S. Haiduc and A. Marcus, "On the use of domain terms in source code," in *Proc. of 16th IEEE International Conference on Program Comprehension*. Amsterdam, the Netherlands: IEEE CS Press, 2008, pp. 113–122.

[24] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[25] S. Holm, "A simple sequentially rejective Bonferroni test procedure," *Scandinavian Journal on Statistics*, vol. 6, pp. 65–70, 1979.

[26] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Hammad, "Assigning change requests to software developers," *Journal of Software Maintenance and Evolution: Research and Practice (JSME)*, vol. 24, no. 1, pp. 3–33, 2012.

[27] E. Keenan, A. Czauderna, G. Leach, J. Cleland-Huang, Y. Shin, E. Moritz, M. Gethers, D. Poshyvanyk, J. Maletic, J. Huffman Hayes *et al.*, "Tracelab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions," in *Proceedings of the 2012 International Conference on Software Engineering*. IEEE Press, 2012, pp. 1375–1378.

[28] J. Kogan, *Introduction to Clustering Large and High-Dimensional Data*. New York, NY, USA: Cambridge University Press, 2007.

[29] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, "What's in a name? a study of identifiers," in *Proc. of 14th IEEE International Conference on Program Comprehension*. Athens, Greece: IEEE CS Press, 2006, pp. 3–12.

[30] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich, "Feature location via information retrieval based filtering of a single scenario execution trace," in *Proc. of 22nd IEEE/ACM International Conference on Automated Software Engineering*. Atlanta, Georgia, USA: ACM Press, 2007.

[31] S. Lohar, S. Amornborvornwong, A. Zisman, and J. Cleland-Huang, "Improving trace accuracy through data-driven configuration and composition of tracing features," in *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*, 2013, pp. 378–388.

[32] Y. S. Maarek, D. M. Berry, and G. E. Kaiser, "An information retrieval approach for automatically constructing software libraries," *IEEE Transactions on Software Engineering*, vol. 17, no. 8, pp. 800–813, 1991.

[33] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proc. of 25th International Conference on Software Engineering*. Portland, Oregon, USA: IEEE CS Press, 2003, pp. 125–135.

[34] A. Marcus, J. I. Maletic, and A. Sergeyev, "Recovery of traceability links between software documentation and source code," *International Journal of Software Engineering and Knowledge Engineering*, vol. 15, no. 5, pp. 811–836, 2005.

[35] A. Michail and D. Notkin, "Assessing software libraries by browsing similar classes, functions and relationships," in *Proc. of 21st International Conference on Software Engineering*. Los Angeles, California, USA: IEEE CS Press, 1999, pp. 463–472.

[36] L. Moreno, G. Bavota, S. Haiduc, M. Di Penta, R. Oliveto, B. Russo, and A. Marcus, "Query-based configuration of text retrieval solutions for software engineering tasks," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, 2015, pp. 567–578.

[37] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "On the equivalence of information retrieval methods for automated traceability link recovery," in *Proc. of the 18th IEEE International Conference on Program Comprehension*, Braga, Portugal, 2010, pp. 68–71.

[38] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "How to effectively use topic models for software

engineering tasks? an approach based on genetic algorithms," in *35th IEEE/ACM International Conference on Software Engineering*, San Francisco, CA, USA, May 18-26, 2013, p. to appear.

[39] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[40] D. Poshyvanyk, Y. Gael-Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Combining probabilistic ranking and latent semantic indexing for feature identification," in *Proc. of 14th IEEE International Conference on Program Comprehension*. Athens, Greece: IEEE CS Press, 2006, pp. 137–148.

[41] D. Poshyvanyk, Y. Gael−Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval," *IEEE Trans. on Softw. Eng.*, vol. 33, no. 6, pp. 420–432, 2007.

[42] D. Poshyvanyk, A. Marcus, R. Ferenc, and T. Gyimóthy, "Using information retrieval based coupling measures for impact analysis," *Empirical Software Engineering*, vol. 14, no. 1, pp. 5–32, 2009.

[43] D. Poshyvanyk and D. Marcus, "Combining formal concept analysis with information retrieval for concept location in source code," in *Proc. of 15th IEEE International Conference on Program Comprehension*. Banff, Alberta, Canada: IEEE CS Press, 2007, pp. 37–48.

[44] D. Poshyvanyk, M. Gethers, and A. Marcus, "Concept location using formal concept analysis and information retrieval," *ACM Trans. Softw. Eng. Methodol.*, vol. 21, no. 4, pp. 23:1–23:34, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2377656.2377660

[45] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2012, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org/

[46] D. Ratiu and F. Deissenboeck, "From reality to programs and (not quite) back again," in *15th IEEE International Conference on Program Comprehension (ICPC'07)*, Banff, Alberta, Canada, June 26-29, 2007, pp. 91–102.

[47] M. Revelle, B. Dit, and D. Poshyvanyk, "Using data fusion and web mining to support feature location in software," in *Proc. of the 18th IEEE International Conference on Program Comprehension*, Braga, Portugal, 2010, pp. 14–23.

[48] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *Proc. of 29th IEEE/ACM International Conference on Software Engineering (ICSE'07)*, Minneapolis, Minnesota, USA, 2007, pp. 499–510.

[49] S. W. Thomas, M. Nagappan, D. Blostein, and A. E. Hassan, "The impact of classifier configuration and classifier combination on bug localization," *IEEE Trans. Software Eng.*, vol. 39, no. 10, pp. 1427–1443, 2013.

[50] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *30th IEEE/ACM International Conference on Software Engineering*, Leipzig, Germany, May 10-18, 2008, pp. 461–470.

[51] Y. Ye and G. Fischer, "Reuse-conducive development environments," *Journal of Automated Software Engineering*, vol. 12, no. 2, pp. 199–235, 2005.