How to Effectively Use Topic Models for Software Engineering Tasks? An Approach Based on Genetic Algorithms

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

***Abstract*—Information Retrieval (IR) methods, and in particular topic models, have recently been used to support essential software engineering (SE) tasks, by enabling software textual retrieval and analysis. In all these approaches, topic models have been used on software artifacts in a similar manner as they were used on natural language documents (e.g., using the same settings and parameters) because the underlying assumption was that source code and natural language documents are similar. However, applying topic models on software data using the same settings as for natural language text did not always produce the expected results.**

Recent research investigated this assumption and showed that source code is much more repetitive and predictable as compared to the natural language text. Our paper builds on this new fundamental finding and proposes a novel solution to adapt, configure and effectively use a topic modeling technique, namely Latent Dirichlet Allocation (LDA), to achieve better (acceptable) performance across various SE tasks. Our paper introduces a novel solution called LDA-GA, which uses Genetic Algorithms (GA) to determine a near-optimal configuration for LDA in the context of three different SE tasks: (1) traceability link recovery, (2) feature location, and (3) software artifact labeling. The results of our empirical studies demonstrate that LDA-GA is able to identify robust LDA configurations, which lead to a higher accuracy on all the datasets for these SE tasks as compared to previously published results, heuristics, and the results of a combinatorial search.

***Index Terms*—Textual Analysis in Software Engineering, Latent Dirichlet Allocation, Genetic Algoritms.**

I. Introduction

A significant amount of research on applying Information Retrieval (IR) methods for analyzing textual information in software artifacts [1] has been conducted in the SE community in recent years. Among the popular and promising IR techniques used, we enumerate Latent Semantic Indexing (LSI) [2] and Latent Dirichlet Allocation (LDA) [3]. The latter is a probabilistic statistical model that estimates distributions of latent topics from textual documents. It assumes that these documents have been generated using the probability distribution of these topics, and that the words in the documents were generated probabilistically in a similar manner.

A number of approaches using LSI and LDA have been proposed to support software engineering tasks: feature location [4], change impact analysis [5], bug localization [6], clone detection [7], traceability link recovery [8], [9], expert developer recommendation [10], code measurement [11], [12], artifact summarization [13], and many others [14], [15], [16]. In all these approaches, LDA and LSI have been used on software artifacts in a similar manner as they were used on natural language documents (i.e., using the same settings, configurations and parameters) because the underlying assumption was that source code (or other software artifacts) and natural language documents exhibit similar properties. More specifically, applying LDA requires setting the number of topics and other parameters specific to the particular LDA implementation. For example, the fast collapsed Gibbs sampling generative model for LDA requires setting the number of iterations n and the Dirichlet distribution parameters α and β [17]. Even though LDA was successfully used in the IR and natural language analysis community, applying it on software data, using the same parameter values used for natural language text, did not always produce the expected results [18]. As in the case of machine learning and optimization techniques, a poor parameter calibration or wrong assumptions about the nature of the data could lead to poor results [19].

Recent research has challenged this assumption and showed that text extracted from source code is much more repetitive and predictable as compared to natural language text [20]. According to recent empirical findings, "*corpus-based statistical language models capture a high level of local regularity in software, even more so than in English*" [20]. This fundamental new research finding explains in part why these fairly sophisticated IR methods showed rather low performance when applied on software data using parameters and configurations that were generally applicable for and tested on natural language corpora.

This paper builds on the finding that text in software artifacts has different properties, as compared to natural language text, thus, we need new solutions for calibrating and configuring LDA and LSI to achieve better (acceptable) performance on software engineering tasks. This paper introduces LDA-GA, a novel solution that uses a Genetic Algorithm (GA) [21] to determine the near-optimal configuration for LDA in the context of three different software engineering tasks, namely (1) traceability link recovery, (2) feature location, and (3) software artifacts labeling.

The contributions of our paper are summarized as follows:

- we introduced LDA-GA, a novel and theoretically sound approach for calibrating LDA on software text corpora using a GA, and we show that it can be applied successfully on three software engineering tasks: traceability link recovery, feature location and software artifact labeling;
- we conducted several experiments to study the performance of LDA configurations based on LDA-GA with those previously reported in the literature; to perform such a comparison we replicated previously published case studies;
- we compared LDA-GA with existing heuristics for calibrating LDA; the empirical results demonstrate that our proposed approach is able to identify LDA configurations that lead to better accuracy as compared to existing heuristics;
- we make publicly available in our online appendix[1] all the data, results and algorithms used in our studies, for replication purposes and to support future studies.

The paper is organized as follows. Section II provides background notions for LDA and overviews its applications to SE problems. Section III describes the LDA-GA approach. Section IV describes the empirical study aimed at applying LDA-GA in the context of traceability link recovery, feature location, and software artifact labeling. Results are reported and discussed in Section V, while Section VI discusses the threats to validity that could have affected our study. Finally, Section VII concludes the paper and outlines directions for future work.

II. Background and Related Work

This section provides (i) background information about LDA, (ii) discussions about its applications to software engineering tasks and (iii) discussions about related approaches aiming at determining the best configuration for LDA.

A. LDA in a Nutshell

Latent Dirichlet Allocation (LDA) [3] is an IR model that allows to fit a generative probabilistic model from the term occurrences in a corpus of documents. Specifically, given a collection of documents, the IR process generates a $m \times n$ term-by-document matrix M, where m is the number of terms occurring in all artifacts, and n is the number of artifacts in the repository. A generic entry w_{ij} of this matrix denotes a measure of the weight (i.e., relevance) of the i^{th} term in the j^{th} document [22]. One of the most used weighting schemas, which we also applied in this work, is the *tf-idf* since it gives more importance to words having high frequency in a document and appearing in a small number of documents [1].

[1]http://www.distat.unimol.it/reports/LDA-GA

Then, the term-by-document matrix is taken as an input by LDA, which identifies the latent variables (topics) hidden in the data and generates as output a $k \times n$ matrix θ, called *topic-by-document matrix*, where k is the number of topics and n is the number of documents. A generic entry θ_{ij} of such a matrix denotes the probability of the j^{th} document to belong to the i^{th} topic. Since typically $k << m$, LDA is mapping the documents from the space of terms (m) into a smaller space of topics (k). The latent topics allow us to cluster them on the basis of their shared topics. More specifically, documents having the same relevant topics are grouped in the same cluster, and documents having different topics belong to different clusters.

LDA requires as input a set of hyper-parameters (i.e., a set of parameters that have a smoothing effect on the topic model generated as output). In this work we used the fast collapsed Gibbs sampling generative model for LDA because it provides the same accuracy as the standard LDA implementation, yet it is much faster [17]. For such an implementation, the set of hyper-parameters are:

- k, which is the number of topics that the latent model should extract from the data. To some extent this is equivalent to the number of clusters in a clustering algorithm;
- n, which denotes the number of Gibbs iterations, where a single iteration of the Gibbs sampler consists of sampling a topic for each word;
- α, which influences the topic distributions per document. A high α value results in a better smoothing of the topics for each document (i.e., the topics are more uniformly distributed for each document);
- β, which affects the term's distribution per topic. A high β value results in a more uniform distribution of terms per topic.

Note that k, α, and β are the parameters of any LDA implementation, while n is an additional parameter required by the Gibbs sampling generative model.

B. LDA Applications to Software Engineering

Some recent applications of LDA to SE tasks operate on models of software artifacts (e.g., source code) rather than directly on those artifacts. Approaches that generate these models require as input a corpus (i.e., a document collection) that represents the software artifacts being analyzed. The corpus is constructed from textual information embedded in the artifacts, including identifiers and comments.

While a number of different SE tasks have been supported using advanced textual retrieval techniques, such as LDA, the common problem remains: the way LDA is commonly configured is based on the assumption that the underlying corpus is composed of natural language text. In our survey of the literature, the following SE tasks have been supported using LDA and all of these papers and approaches used ad-hoc heuristics to configure LDA, perhaps resulting in sub-optimal performance in virtually all the cases: feature location [23], bug localization [6], impact analysis [24], source code labeling [13], aspect identification [14], expert identification [25], software traceability [9], [26], test case prioritization [27], and evolution analysis [28], [16].

C. Approaches for Estimating the Parameters of LDA

Finding an LDA configuration that provides the best performance is not a trivial task. Some heuristics have been proposed [29], [30]; however, these approaches focus only on identifying the number of topics that would result in the best performance of a task, while ignoring all the other parameters that are required to apply LDA in practice. Moreover, such approaches have not been evaluated on real SE applications or have been defined for natural language documents only, thus, they may not be applicable for software corpora.

One such technique is based on a heuristic 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 [29]. However, the utility of this heuristic was not evaluated in the context of specific SE tasks.

On a more theoretical side, a non-parametric extension of LDA called Hierarchical Dirichlet Processes [31] tries to infer the optimal number of topics automatically from the input data. Griffiths and Steyvers [30] proposed 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 $P(z|w,T)$, (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 (i.e., $P(w|T)$), and (iv) choosing the topic with the maximum likelihood. In their approach, the hyper-parameters α and β are fixed, and only the number of topics is varied, which in practice, is not enough to properly calibrate LDA. In our approach, we vary all the parameters (i.e., k, n, α and β), to find a (near) optimal configuration for LDA.

III. Finding a (Near) Optimal LDA Configuration for Software

As explained in Section II, LDA—and in particular its implementation based on fast collapsed Gibbs sampling generative model—requires the calibration of four parameters, k, n, α, and β. Without a proper calibration, or with an ad-hoc calibration of these parameters, LDA's performance may be sub-optimal. Finding the best configuration of these parameters poses two problems. Firstly, we need a measure that can be used to assess the performances of LDA before applying it to a specific task (e.g., traceability link recovery). This measure should be independent from the supported SE task. In other words, we cannot simply train an LDA model on the data for one particular task, since obtaining such data means solving the task. For example, for traceability link recovery, if we identify all the links to assess the quality of the LDA model for extracting the links themselves, then there is no need to have an LDA-based model to recover these links anymore. In other words, we need to build such a model on raw data (e.g., source code and documentation) without having additional information about the links. Secondly, we need an efficient way to find the best configuration of parameters, as an exhaustive analysis of all possible combinations is impractical due to (i) the combinatorial nature of the problem (i.e., the large number of possible configuration values for the LDA parameters), as well as (ii) the large amount of computational time required for even such a configuration. In the next section we present our approach that addresses these problems, called LDA-GA, which is able to find a near-optimal configuration for the parameters of LDA.

A. Assessing the Quality of an LDA Configuration

LDA can be considered as a topic-based clustering technique, which can be used to cluster documents in the topics space using the similarities between their topics distributions. Our conjecture is that there is a strong relationship between the performances obtained by LDA on software corpora and the quality of clusters produced by LDA. Thus, measuring the quality of the produced clusters could provide some insights into the accuracy of LDA when applied to software engineering tasks. Indeed, if the quality of the clusters produced by LDA is poor, it means that LDA was not able to correctly extract the dominant topics from the software corpus, because the documents, which are more similar to each other, are assigned to different clusters (i.e., LDA assigns different dominant topics to neighboring documents).

In our paper, we use the concept of a dominant topic to derive the textual clustering generated by a particular LDA configuration applied on a term-by-document matrix. Formally, the concept of a dominant topic can be defined as follows:

Definition 1. *Let θ be the topic-by-document matrix generated by a particular LDA configuration $P = [k, n, \alpha, \beta]$. A generic document d_j has a dominant topic t_i, if and only if $\theta_{i,j} = max\{\theta_{h,j},\ h = 1 \dots k\}$.*

Starting from the definition of the dominant topic, we can formalize how LDA clusters documents within the topic space (the number of clusters is equal to the number of topics) as follows:

Definition 2. *Let θ be the topic-by-document matrix generated by a particular LDA configuration $P = [k, n, \alpha, \beta]$. A generic document d_j belongs to the i^{th} cluster, if and only if t_i is the dominant topic of d_j.*

Thus, we can define a cluster as a set of documents in which each document is closer (i.e., shares the same dominant topic) to every other document in the cluster, and it is further from any other document from the other clusters. It is worth noting that the concept of a dominant topic is specific to software documents only. Collections of natural language documents are usually heterogeneous, meaning that documents can contain information related to multiple topics. In source code artifacts, heterogeneity is not always present, especially when considering single classes. More specifically, a class is a

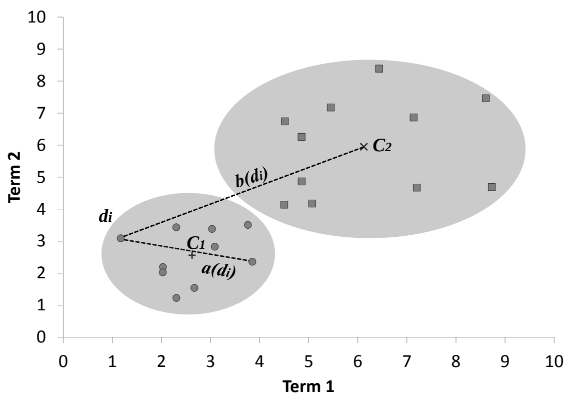


Fig. 1. Example of the Silhouette coefficient.

crisp abstraction of a domain/solution object, and should have a few, clear responsibilities. Hence, software documents should be clustered considering only the dominant topic, assuming that each document is related to only one specific topic.

Different LDA configurations provide different clustering models of the documents. However, not all clustering models that can be obtained by configuring LDA are good. There are two basic ways to evaluate the quality of a clustering structure: *internal criteria*, based on similarity/dissimilarity between different clusters and *external criteria*, which uses additional and external information (e.g., using judgement provided by users) [32]. Since the internal criterion does not require any manual effort and it is not software engineering task dependent, in our paper we use the *internal criteria* for measuring the quality of clusters. More specifically, we use two types of internal quality metrics: *cohesion* (similarity), which determines how closely related the documents in a cluster are, and *separation* (dissimilarity), which determines how distinct (or well-separated) a cluster is from other clusters[32]. Since these two metrics are contrasting each other, we use a popular method for combining both cohesion and separation in only one scalar value, called *Silhouette coefficient* [32]. The Silhouette coefficient is computed for each document using the concept of centroids of clusters. Formally, let C be a cluster; its centroid is equal to the mean vector of all documents belonging to C: $Centroid(C) = \sum_{d_i \in C} d_i/|C|$.

Starting from the definition of centroids, the computation of the Silhouette coefficient consists of the following three steps:

1) For document d_i, calculate the maximum distance from d_i to the other documents in its cluster. We call this value $a(d_i)$.
2) For document d_i, calculate the minimum distance from d_i to the centroids of the clusters not containing d_i. We call this value $b(d_i)$.
3) For document d_i, the Silhouette coefficient $s(d_i)$ is:

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

The value of the Silhouette coefficient ranges between -1 and 1. A negative value is undesirable, because it corresponds to the case in which $a(d_i) > b(d_i)$, i.e., the maximum distance to other documents in the cluster is greater then the minimum distance to other documents in other clusters. For measuring the distance between documents we used the Euclidean distance, since it is one of the most commonly used distance functions for data clustering [32]. Fig. 1 provides a graphical interpretation of the Silhouette coefficient computed for a document d_i. In particular, it represents an example of a good Silhouette coefficient, since d_i is close to the furthest document situated in its cluster, and far from the centroid of the nearest cluster. In the end, the overall measure of the quality of a clustering $C = \{C_1, \ldots, C_k\}$ can be obtained by computing the mean Silhouette coefficient of all documents. Let $C = \{C_1, \ldots, C_k\}$ be the clustering obtained using a particular LDA configuration, and let M be an $m \times n$ term-by-document matrix. The mean Silhouette coefficient is computed as:

$$s(C) = \frac{1}{n}\sum_{i=1}^{n} s(d_i)$$

In this paper, we used the mean Silhouette coefficient as the measure for predicting the accuracy of LDA in the context of specific software engineering tasks.

B. Finding a (Near) Optimal LDA Configuration

Based on the conjecture that the higher the clustering quality produced by LDA, the higher the accuracy of LDA when used for software engineering tasks, we present an approach to efficiently identify the LDA configuration $P = [k, n, \alpha, \beta]$ that maximizes the overall quality (measured using the mean Silhouette coefficient) of the clustering produced by LDA. For solving such an optimization problem we applied Genetic Algorithms (GA) which is a stochastic search technique based on the mechanism of a natural selection and natural genetics. Since the introduction of GA by Holland [21] in the 1970s, this algorithm has been used in a wide range of applications where optimization is required and finding an exact solution is NP-Hard. The advantage of GA with respect to the other search algorithm is its intrinsic parallelism, i.e., having multiple solutions (individuals) evolving in parallel to explore different parts of the search space.

The GA search starts with a random population of solutions, where each individual (i.e., *chromosome*) from the population represents a solution of the optimization problem. The population evolves 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, for 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*. More details about GA can be found in a book by Goldberg [33].

In our paper, we used a simple GA [33] with elitism of two individuals (i.e., the two best individuals are kept alive across generations). Individuals (solutions) are represented as an array with four floats, where each element represents k, n, α, and β, respectively. Thus, an individual (or chromosome) is a particular LDA configuration and the population is represented by a set of different LDA configurations. The selection operator is the Roulette wheel selection, which assigns to the individuals with higher fitness a higher chances to be selected. The crossover operator is the arithmetic crossover, that creates new individuals by performing a linear combination—with random coefficients—of the two parents. The mutation operator is the uniform mutation, which randomly changes one of the genes (i.e., one of the four LDA parameter values) of an individual, with a different parameter value within a specified range. The fitness function that drives the GA evolution is the Silhouette coefficient described in Section III-A. Our GA approach can be briefly summarized as (i) generating LDA configurations, (ii) using them to cluster documents, (iii) evaluating the cluster quality using the Silhouette coefficient, and (iv) using that value to drive the GA evolution.

IV. Empirical Study Definition

This section describes the design of the empirical studies that we conducted in order to evaluate LDA-GA in the context of three software engineering tasks. The studies aim at answering the following research questions:

- **RQ**$_1$: *What is the impact of the configuration parameters on LDA's performance in the context of software engineering tasks?* This research question aims at justifying the need for an automatic approach that calibrates LDA's settings when LDA is applied to support SE tasks. For this purpose, we analyzed a large number of LDA configurations for three software engineering tasks. The presence of a high variability in LDA's performances indicates that, without a proper calibration, such a technique risks being severely under-utilized.
- **RQ**$_2$: *Does LDA-GA, our proposed GA-based approach, enable effective use of LDA in software engineering tasks?* This research question is the main focus of our study, and it is aimed at analyzing the ability of LDA-GA to find an appropriate configuration for LDA, which is able to produce good results for specific software engineering tasks.

We address both research questions in three different scenarios, representative of SE tasks that can be supported by LDA: traceability link recovery, feature location, and software artifact labeling. LDA was previously used in some of these tasks [9], [28], [13]. For our data and results please visit our online appendix.

A. Scenario I: Traceability Links Recovery

In this scenario, we used LDA to recover links between documentation artifacts (e.g., use cases) and code classes. The experiment has been conducted on software repositories from two projects, EasyClinic and eTour. EasyClinic is a system used to manage a doctor's office, while eTour is an electronic touristic guide. Both systems were developed by the final year Master's students at the University of Salerno (Italy). The documentation, source code identifiers, and comments for both systems are written in Italian. The top part of Table I reports the characteristics of the considered software systems in terms of type, number of source and target artifacts, as well as Kilo Lines of Code (KLOC). The table also reports the number of correct links between the source and target artifacts. These correct links, which are derived from the traceability matrix provided by the original developers, are used as an *oracle* to evaluate the accuracy of the proposed traceability recovery method.

TABLE I
CHARACTERISTICS OF THE SYSTEMS USED IN THE THREE SCENARIOS.

Scenario I - Traceability Link Recovery

System	KLOC	Source Artifacts (#)	Target Artifacts (#)	Correct links
EasyClinic	20	UC (30)	CC (47)	93
eTour	45	UC (58)	CC (174)	366

UC: Use case, CC: Code class

Scenario II - Feature Location

System	KLOC	Classes	Methods	Features
jEdit	104	503	6,413	150
ArgoUML	149	1,439	11,000	91

Scenario III - Software Artifact Labeling

System	KLOC	Classes	Sampled classes
JHotDraw	29	275	10
eXVantage	28	348	10

To address **RQ**$_1$, we compared the accuracy of recovering traceability links using different configurations for LDA. Specifically, we varied the number of topics from 10 to 100 with step 10 on EasyClinic, and from 10 to 200 with step 10 on eTour. We varied α and β from 0 to 1 with 0.1 increments, and we exercised all possible combinations of such values. We fixed the number of iterations to 500, which resulted to be a sufficient number of iterations for the model to converge. Thus, the total number of trials performed on EasyClinic and eTour were 1,000 and 2,000, respectively. Clearly, although combinatorial, such an analysis is not exhaustive, as it considers a discrete set of parameter values and combinations. For **RQ**$_2$, we compared the accuracy achieved by LDA when the configuration is determined using LDA-GA with (i) the best accuracy achieved by LDA (determined when answering RQ$_1$) and (ii) the accuracy achieved by LDA on the same system in the previously published studies where an "ad-hoc" configuration was used [26]. While the former comparison is more of a sanity check aimed at analyzing the effectiveness of the GA in finding a near-optimal solution, the latter comparison was aimed at analyzing to what extent LDA-GA is able to enrich the effectiveness and usefulness of LDA in the context of traceability link recovery when properly calibrated.

When addressing **RQ**$_1$, we evaluated LDA's recovery accuracy using the average precision metric [22], which returns a single value for each ranked list of candidate links provided. For **RQ**$_2$, we used two well-known IR metrics: precision and recall [22]. The precision values achieved for different configurations (over different levels of recall) 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 [34]. This procedure sorts the p-values resulting from n tests in ascending order, multiplying the smallest by n, the next by $n-1$, and so on.

B. Scenario II: Feature Location

In this scenario, we used LDA to locate features within the textual corpus of source code. The context of this scenario is represented by two software systems, *jEdit v4.3* and *ArgoUML v0.22*. jEdit is an open-source text editor for programmers, while ArgoUML is a well known UML editor. Table I reports the characteristics of the considered software systems in terms of number of classes, number of methods, as well as KLOC and the number of features to be located. These software systems have been used in previous studies on feature location [23], [35]. For more information about the datasets refer to [4].

To answer **RQ**$_1$, we compared the effectiveness measure of LDA using different configurations. Specifically, we varied the number of topics from 50 to 500 with step 50 for both ArgoUML and jEdit. We varied α and β from 0 to 1 with 0.1 increments. Similarly to the traceability task, we fixed the number of iterations to 500. We exercised all possible combinations of such values. Thus, the total number of trials performed on both software systems consisted of 1,000 different LDA combinations. For **RQ**$_2$, similarly to the previous scenario, we compared the performance achieved by LDA-GA with (i) the best performance achieved by LDA when answering **RQ**$_1$ and (ii) the performance obtained by LDA using the *source locality* heuristic proposed by Grant and Cordy for the feature location task [29]. The performance of LDA in this scenario was analyzed using the *effectiveness measure (EM)* [36]. Given a feature of interest, this measure estimates the number of methods a developer needs to inspect before finding a method relevant to that feature (the list of methods are ranked by their similarity to the description of the feature). A lower value for the EM indicates less effort (i.e., fewer methods to analyze before finding a relevant one). The EM computed for different configurations on different queries (i.e., feature descriptions) were then pairwise-compared using a Wilcoxon rank sum test, similarly to the evaluation from Scenario I and, also in this case, the p-values were adjusted using Holm's procedure.

C. Scenario III: Software Artifact Labeling

In this scenario, we used LDA to automatically "label" source code classes using representative words. Specifically, we extracted topics from a single class (using LDA), and then we ranked all the words characterizing the extracted topics according to their probability in the obtained topic distribution. The top 10 words belonging to the topic with the highest probability in the obtained topic distribution were then used to label the class [13].

The study was conducted on 10 classes from JHotDraw and 10 classes from eXVantage. The former is an open-source drawing tool, and the latter is a novel testing and generation tool. Their characteristics are summarized in the bottom part of Table I. For the sampled classes, we had user-generated labels from a previously published work [13], and these represented our "ideal" labels. After obtaining the LDA labels we compared them to the user-generated ones and computed the overlap between them. The overlap was measured using the asymmetric Jaccard measure [22]. Formally, let $K(C_i) = \{t_1, \ldots, t_m\}$ and $K_{m_i}(C_i) = \{t_1, \ldots, t_h\}$ be the sets of keywords identified by subjects and the technique m_i, respectively, to label the class C_i. The overlap was computed as follows:

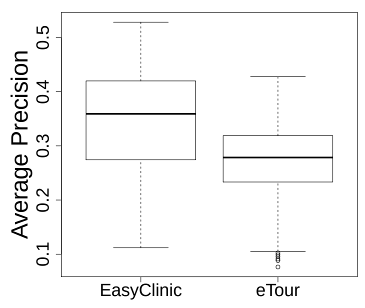
$$overlap_{m_i}(C_i) = \frac{|K(C_i) \cap K_{m_i}(C_i)|}{K_{m_i}(C_i)}$$

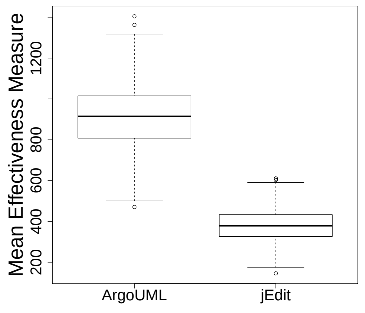
Note that the size of $K(C_i)$ might be different from the size of $K_{m_i}(C_i)$. In particular, while the number of keywords identified by LDA is always 10 (by construction we set $h = 10$), the number of keywords identified by subjects could be more or less than 10 (generally it is 10, but there are few cases where the number is different). For this reason, we decided to use the asymmetric Jaccard to avoid penalizing too much the automatic method when the size of $K(C_i)$ is less than 10.

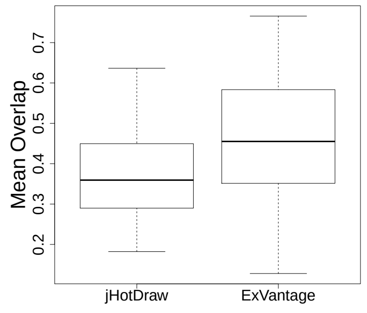
Also in this scenario, in order to address **RQ**$_1$ we compared the recovery accuracy of LDA using different settings. Specifically, we varied the number of topics from 10 to 50 with step 10 for both JHotDraw and eXVantage. As for α and β, we varied them between 0 and 1 by increments of 0.1. We fixed the number of iterations to 500 as in the previous two tasks. We exercised all possible combinations of such values. Thus, the total number of trials performed on JHotDraw and eXVantage was 500 on both systems. For **RQ**$_2$, we compared the accuracy achieved by LDA-GA with (i) the best accuracy achieved by LDA while iterating through the parameters and (ii) the accuracy achieved by LDA reported by De Lucia *et al.* [13].

D. LDA-GA Settings and Implementation

The LDA-GA has been implemented in `R` [37] using the `topicmodels` and `GA` libraries. The former library provides a set of routines for computing the fast collapsed Gibbs sampling generative model for LDA, while the latter is a collection of general purpose functions that provide a flexible set of tools for applying a wide range of GA methods. For GA, we used the following settings: a crossover probability of 0.6, a mutation probability of 0.01, a population of 100 individuals, and an elitism of 2 individuals. As a stopping criterion for the GA, we terminated the evolution if the best results achieved did not improve for 10 generations; otherwise we stopped after 100 generations. All the settings have been calibrated using a trial-and-error procedure, and some of them







(a) Traceability

(b) Feature Location

(c) Labeling

Fig. 2. Variability of performance achieved by LDA configurations for (a) traceability link recovery, (b) feature location, and (c) labeling.

(i.e., elitism size, crossover and mutation probabilities) were the values commonly used in the community. To account for GA's randomness, for each experiment we performed 30 GA runs, and then we took the configuration achieving the median final value of the fitness function (i.e., of the Silhouette coefficient).

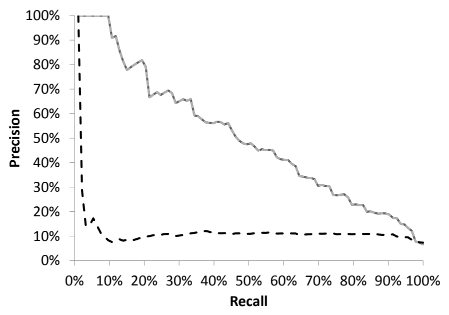
V. Empirical Study Results

This section discusses the results of our experiments conducted in order to answer the research questions stated in Section IV. We report the results for each LDA application scenario.

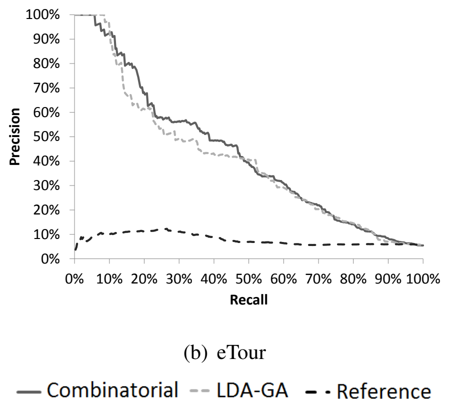
A. Scenario I: Traceability Link Recovery

As for **RQ**$_1$, Fig. 2-a shows boxplots summarizing the average precision values obtained using the 1,000 and 2,000 different LDA configurations (described in Section IV) on EasyClinic and eTour, respectively. We used these boxplots to highlight the variability of the average precision values across different configurations. As shown, the variability of LDA's performance is relatively high: the average precision ranges between 11% and 55% on EasyClinic and between 7% and 43% on eTour. For EasyClinic, more than 75% of the different LDA configurations obtained an average precision lower than 45% (see first three quartiles in Fig. 2-a). Moreover, only a small percentage of the configurations executed in the combinatorial search (about 3.6%) obtained an average precision greater than 50%. In the end, only one of them achieved the highest value, 52%. Similarly for eTour, the configurations placed in the first three quartiles (about 75% of the set) obtained an average precision lower than 40%, while less than 1% of the total amount of executed configurations in the combinatorial search (2,000 configurations) achieved an average precision greater than the 40%. Only one configuration achieved the highest average precision (47%).

In summary, for **RQ**$_1$ we can assert that for traceability recovery, LDA shows high variability. Thus, LDA's efficiency for establishing links between software artifacts depends on the particular configuration $P = [n, k, \alpha, \beta]$ used to derive latent topics. Indeed, "bad" configurations can produce poor



(a) EasyClinic



(b) eTour

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

results while "optimal" configurations (which represent a small portion of all possible LDA configurations) can lead to very good results.

Regarding **RQ**$_2$, Fig. 3 reports the precision/recall graphs obtained by LDA using (i) the best configuration across 1,000 and 2,000 different configurations executed in the combinatorial search; (ii) the configuration identified by LDA-GA; and (iii) an "ad-hoc" configuration used in a previous study where LDA was used on the same repositories [18]. For both EasyClinic and eTour, LDA-GA was able to obtain a recovery accuracy close to the accuracy achieved by the optimal

TABLE II
RESULTS OF THE WILCOXON TEST FOR TRACEABILITY RECOVERY.

	EasyClinic	eTour
LDA-GA < Combinatorial	1	1
LDA-GA < Oliveto *et al.* [18]	< **0.01**	< **0.01**
Combinatorial < Oliveto *et al.* [18]	< **0.01**	< **0.01**
Combinatorial < LDA-GA	1	< **0.01**

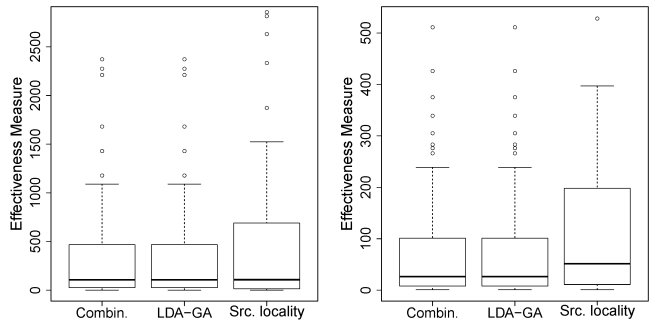
configuration across 1,000 and 2,000 different configurations executed in the combinatorial search. In particular, for EasyClinic LDA-GA returned exactly the configuration identified by the combinatorial search (i.e., the two curves are perfectly overlapped) while on eTour the two curves are comparable. Moreover, the average precision achieved by the configuration provided by LDA-GA is about 41%, which is comparable with the average precision achieved with the the optimal configuration, which is about 43% (only a small difference of 2%). Among 2,000 different configurations tried for the combinatorial search, only five configurations obtained an average precision comparable or greater than the one achieved by LDA-GA, i.e., the configurations obtained by LDA-GA belong to the 99% percentile for the distribution reported in Fig. 2-a. Finally, comparing the performance achieved by LDA-GA with the performance reached by other LDA configurations used in previous work [18], we can observe that the improvement is very substantial for both software systems.

Table II reports the results of the Wilcoxon test (i.e., the adjusted p-values) for all combinations of the techniques (statistically significant results are highlighted in bold face). As we can see, there is no statistically significant difference between the performance obtained by LDA-GA and the combinatorial search for EasyClinic. However, for eTour the combinatorial search performs significantly better than LDA-GA. However, considering the precision/recall graph reported in Fig. 3, we can observe that the difference is relatively small.

B. Scenario II: Feature Location

Fig. 2-b shows the boxplots summarizing the variability of the average effectiveness measure (EM) values obtained using 1,000 different LDA configurations, as explained in Section IV. As in the previous task, the feature location results show high variability in their EM, which ranges between 472 and 1,416 for ArgoUML and between 145 and 600 for jEdit. For ArgoUML, we observed that more than 90% of different configurations produced an average EM ranging between 600 and 1,200, while only a small percentage (about 3%) produced an optimal average EM lower than 600. Within this small number of optimal configurations only one configuration obtains the lowest (i.e., the best) EM of 472. Similarly, for jEdit, 95% of different configurations produced an average EM that ranges between 200 and 1,600, while only one achieved the smallest average EM of 145. These results for $\mathbf{RQ}_1$ suggest that without a proper calibration, the performance of LDA risks of being unsatisfactory.

For $\mathbf{RQ}_2$, Fig. 4-a shows boxplots for ArgoUML of the EM values achieved by three different configurations: (i) the best configuration obtained by a combinatorial search across 1,000 different LDA configurations (combinatorial search); (ii) the configuration obtained using LDA-GA; and (iii) the best configuration obtained using the source locality heuristic [29]. First, we can note that the configuration obtained via LDA-GA is exactly the same as the one obtained from the combinatorial search, thus LDA-GA was able to find the best configuration (i.e., with the lowest average EM). Comparing the performance of LDA-GA with the source locality heuristic, we can observe that for the first two quartiles, there is no clear difference (the median values are 107 and 108 for LDA-GA and source locality heuristic respectively). Considering the third and fourth quartiles, the difference becomes substantial: the third quartile is 467 for LDA-GA and 689 for the previous heuristic, while for the fourth quartiles we obtained 4,603 for LDA-GA and 7,697 for source locality heuristic. Overall, LDA-GA reached an average EM equal to 473, as opposed to EM equal to 707 obtained using the source locality heuristic.



(a) ArgoUML (b) jEdit

Fig. 4. Box plots of the effectiveness measure for feature location for ArgoUML and jEdit.

TABLE III
RESULTS OF THE WILCOXON TEST ON FEATURE LOCATION PERFORMANCES.

	jEdit	ArgoUML
LDA-GA < Combinatorial	0.09	1
LDA-GA < Source Locality Heuristic [29]	**0.02**	**0.02**
Combinatorial < Source Locality Heuristic [29]	**0.02**	**0.02**

Table III reports the results of the Wilcoxon test (i.e., the adjusted p-values) for all combinations of the techniques (statistically significant results are shown in bold face). As we can see, there is no statistical difference between the performance obtained by LDA-GA and the combinatorial search. Based on the results of the statistical tests, we can assert that LDA-GA is able to find the optimal or the near-optimal configurations. Moreover, LDA-GA significantly outperforms the previously published source locality heuristic ($p\text{-value} < 0.02$).

C. Scenario III: Software Artifact Labeling

For $\mathbf{RQ}_1$, Fig. 2-c shows boxplots for the average percentage overlap (AO) values obtained using 500 different LDA configurations, as explained in Section IV. Even if in this case the corpus of documents (the total number of classes and the vocabulary size) is really small, as compared to the

TABLE IV
Average Overlap between automatic and manual labeling.

exVantage					
	LDA		**De Lucia et al. [13]**		
	LDA-GA	**Combinatorial**	**n = M**	**n = M/2**	**n = 2**
Max	**100%**	**100%**	100%	100%	100%
3rd Quartile	**95%**	**95%**	71%	70%	69%
Median	67%	**70%**	59%	60%	54%
2nd Quartile	60%	**67%**	34%	50%	41%
Min	**50%**	**50%**	0%	0%	40%
Mean	74%	77%	52%	56%	60%
St. Deviation	19%	17%	31%	34%	23%

JHotDraw					
	LDA		**De Lucia et al. [13]**		
	LDA-GA	**Combinatorial**	**n = M**	**n = M/2**	**n = 2**
Max	**100%**	**100%**	100%	100%	100%
3 Quartile	81%	**82%**	73%	70%	66%
Median	71%	**75%**	65%	61%	56%
2 Quartile	47%	**50%**	46%	45%	41%
Min	14%	14%	0%	38%	**29%**
Mean	65%	66%	59%	60%	59%
St. Deviation	28%	26%	28%	20%	24%

size of the repository considered for the other tasks, LDA also shows a high variability of performances, ranging between 18% and 66% on JHotDraw, and between 13% and 77% on ExVantage. For JHotDraw, it can be noted how, more than 72% of the different configurations obtained an AO value ranging between 25% and 55%, while only a small percentage (about 1%) obtains an optimal AO greater than 60%. Within this small number of optimal configurations, only one achieves the highest AO of 64%. Similarly, for ExVantage the majority (about 79%) of the different configurations obtained an AO ranging between 10% and 70%, while only one configuration achieved the highest AO of 77%.

For $\mathbf{RQ}_2$, Table IV reports the statistics of the overlap between the user-based labeling and the automatic labeling obtained using (i) LDA-GA; (ii) the best configuration achieved using the combinatorial search, i.e., the configuration which has the higher percentage overlap among 500 different configurations; and (iii) the LDA configuration used in the previous work [13] for the same task. For both systems, LDA-GA obtains a percentage overlap with the user labeling that is close to the combinatorial search, with a difference from the best LDA configuration (obtained by the combinatorial search) of about 3% for ExVantage and 1% for JHotDraw. For ExVantage, among the 500 different LDA configurations computed in the combinatorial search, only 12 configurations have an average overlap greater or equal to 74.33%. We can also observe that there are only small differences for the median and second quartile between LDA-GA and the global optimum, while for the other quartiles there is no difference. Similarly, among 500 different configurations evaluated for JHotDraw, only one configuration is comparable with LDA-GA. By comparing the quartile values obtained for JHotDraw, we can note that the difference between LDA-GA and the combinatorial search optimum is about 2%-3% on average. Finally, we can observe how the performances of LDA configured using LDA-GA are significantly better than those reported in the previous work [13] (where α and β were set to default of $50/k$ and 0.1 respectively). For ExVantage we obtain an improvement in terms of mean overlap of about 14-20%, while for JHotDraw we get an improvement of about 5-6%.

VI. Threats to Validity

Threats to *construct validity* concern the relationship between theory and observation. For tasks such as traceability link recovery and feature location, we used well-established metrics (i.e., precision, recall and effectiveness) and oracles used in previous studies [23], [35], [38], [26], thereby ensuring that the error-proneness is limited. For the labeling task, we compared LDA-based labeling with a user-generated labeling, using, again, the dataset previously verified and published [13].

Threats to *internal validity* can be 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. We also observed that the configuration that we obtained did not substantially vary across GA runs.

Threats to *conclusion validity* concern the relationship between treatment and outcome. Wherever appropriate, we used non-parametric statistical tests (the Wilcoxon test rank sum test in particular) to support our claims.

Threats to *external validity* concern the generalization of our results. Firstly, it is highly desirable to replicate the studies carried out on three scenarios on other datasets. Secondly, although the proposed approach can be applied in principle to other LDA-based solutions to support SE tasks, specific studies are needed to evaluate their feasibility and performances.

VII. Conclusion and Future Work

In this paper we proposed LDA-GA, an approach based on Genetic Algorithms that determines the near-optimal configuration for LDA in the context of three important software engineering tasks, namely (1) traceability link recovery, (2) feature location, and (3) software artifact labeling. We also conducted several experiments to study the performance of LDA configurations based on LDA-GA with those previously reported in the literature (i.e., existing heuristics for calibrating LDA) and a combinatorial search. The results obtained indicate that (i) applying LDA to software engineering tasks requires a careful calibration due to its high sensitivity to different parameter settings, that (ii) LDA-GA is able to identify LDA configurations that lead to higher accuracy as compared to alternative heuristics, and that (iii) its results are comparable to the best results obtained from the combinatorial search.

Overall, our empirical results warn the researchers about the dangers of ad-hoc calibration of LDA on software corpora, as was predominantly done in the SE research community, or using the same settings and parameters applicable only to natural language texts. Without a sound calibration mechanism for LDA on software data, which might require using approaches such as the one proposed in this paper, the potential of such a rigorous statistical method as LDA can be seriously undermined, as shown in our empirical study.

Future work will be devoted to corroborating the results reported in this paper on other datasets. We also plan to apply LDA-GA on other SE tasks that rely on text analysis using topic models.

VIII. Acknowledgments

This work is supported in part by NSF CCF-1016868, CNS-0959924, and NSF CCF-1218129 awards. Any opinions, findings and conclusions expressed here are the authors' and do not necessarily reflect those of the sponsors.

References

[1] D. Binkley and D. Lawrie, "Information retrieval applications in software maintenance and evolution," *Encycl. of Softw. Engineering*, 2009.

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

[3] 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.

[4] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: A taxonomy and survey," *Journal of Software: Evolution and Process (JSEP)*, doi: 10.1002/smr.567/, 2012 (to appear).

[5] 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.

[6] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Bug localization using Latent Dirichlet Allocation," *Information and Software Technology*, vol. 52, no. 9, pp. 972–990, 2010.

[7] R. Tairas and J. Gray, "An Information Retrieval process to aid in the analysis of code clones," *Empirical Software Engineering*, vol. 14, no. 1, pp. 33–56, 2009.

[8] 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, 2003, pp. 125–135.

[9] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *Proc. of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10)*. Cape Town, South Africa, May 2-8: ACM, 2010, pp. 95–104.

[10] 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.

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

[12] Y. Liu, D. Poshyvanyk, R. Ferenc, T. Gyimóthy, and N. Chrisochoides, "Modeling class cohesion as mixtures of latent topics," in *25th IEEE International Conference on Software Maintenance (ICSM'09)*, Edmonton, Alberta, Canada, September 20-26, 2009, pp. 233–242.

[13] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Using ir methods for labeling source code artifacts: Is it worthwhile?" in *IEEE 20th International Conference on Program Comprehension (ICPC'12)*, Passau, Germany, June 11-13, 2012, pp. 193–202.

[14] P. Baldi, C. V. Lopes, E. Linstead, and S. K. Bajracharya, "A theory of aspects as latent topics," in *Proc. of the 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'08)*, Nashville, TN, USA, 2008, pp. 543–562.

[15] 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.

[16] A. J. Hindle, M. W. Godfrey, and R. C. Holt, "What's hot and what's not: Windowing developer topic analysis," in *Proc. of the 25th IEEE International Conference on Software Maintenance (ICSM'09), Edmonton, Canada, September 20-26*, 2009.

[17] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling, "Fast collapsed gibbs sampling for latent dirichlet allocation," in *Proc. of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 569–577.

[18] 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 (ICPC'10)*, Braga, Portugal, 2010, pp. 68–71.

[19] A. Arcuri and G. Fraser, "On parameter tuning in search based software engineering," in *Search Based Software Engineering - Third International Symposium, SSBSE 2011, Szeged, Hungary, September 10-12, 2011. Proc.* Springer, 2011, pp. 33–47.

[20] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. T. Devanbu, "On the naturalness of software," in *Proc. of the 34th IEEE/ACM International Conference on Software Engineering (ICSE'12), Zurich, Switzerland, June 2-9*, 2012, pp. 837–847.

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

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

[23] L. Biggers, C. Bocovich, R. Capshaw, B. Eddy, L. Etzkorn, and N. Kraft, "Configuring Latent Dirichlet Allocation based feature location," *Empirical Software Engineering (EMSE)*, p. to appear, 2012.

[24] A. Nguyen, T. Nguyen, J. Al-Kofahi, H. Nguyen, and T. Nguyen, "A topic-based approach for narrowing the search space of buggy files from a bug report," in *Proc. of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'11)*. Lawrence, Kansas, USA, November 6-10: IEEE, 2011, pp. 263–272.

[25] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi, "Mining eclipse developer contributions via author-topic models," in *Proc. of 4th International Workshop on Mining Software Repositories*. Minneapolis, Minnesota, USA: IEEE CS Press, 2007, pp. 30–33.

[26] 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.

[27] S. W. Thomas, H. Hemmati, A. E. Hassan, and D. Blostein, "Static test case prioritization using topic models," 2013, forthcoming in *Empirical Software Engineering*.

[28] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Validating the use of topic models for software evolution," in *Tenth IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2010, Timisoara, Romania, 12-13 September 2010*. IEEE Computer Society, 2010, pp. 55–64.

[29] 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.

[30] 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.

[31] Y. Teh, M. Jordan, M. Beal, and D. Blei, "Hierarchical Dirichlet processes," *Journal of the American Statistical Association*, vol. 101, no. 476, pp. 1566–1581, 2006.

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

[33] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Longman Publishing Co., 1989.

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

[35] 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.

[36] 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.

[37] 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/

[38] M. Gethers and D. Poshyvanyk, "Using relational topic models to capture coupling among classes in object-oriented software systems," in *26th IEEE International Conference on Software Maintenance (ICSM 2010), September 12-18, 2010, Timisoara, Romania*, 2010, pp. 1–10.