Configuring Topic Models for Software Engineering Tasks in TraceLab

Bogdan Dit[1], Annibale Panichella[2], Evan Moritz[1], Rocco Oliveto[3], Massimiliano Di Penta[4], Denys Poshyvanyk[1], Andrea De Lucia[2]

[1]The College of William and Mary, Williamsburg, VA, USA
[2]University of Salerno, Fisciano (SA), Italy
[3]University of Molise, Pesche (IS), Italy
[4]University of Sannio, Benevento, Italy

***Abstract*—A number of approaches in traceability link recovery and other software engineering tasks incorporate topic models, such as Latent Dirichlet Allocation (LDA). Although in theory these topic models can produce very good results if they are configured properly, in reality their potential may be undermined by improper calibration of their parameters (e.g., number of topics, hyper-parameters), which could potentially lead to sub-optimal results. In our previous work we addressed this issue and proposed LDA-GA, an approach that uses Genetic Algorithms (GA) to find a near-optimal configuration of parameters for LDA, which was shown to produce superior results for traceability link recovery and other tasks than reported ad-hoc configurations. LDA-GA works by optimizing the coherence of topics produced by LDA for a given dataset. In this paper, we instantiate LDA-GA as a TraceLab experiment, making publicly available all the implemented components, the datasets and the results from our previous work. In addition, we provide guidelines on how to extend our LDA-GA approach to other IR techniques and other software engineering tasks using existing TraceLab components.**

***Index Terms*—Configurable, TraceLab, experiments, LDA, genetic algorithm, traceability**

I. Introduction

The recent years showed a significant progress in software engineering (SE) research community on applying Information Retrieval (IR) techniques. One of these techniques, called Latent Dirichlet Allocation (LDA) [1], is a topic modeling technique that can identify latent concepts found in the unstructured textual information of software artifacts. More specifically, LDA uses a probabilistic statistical model to extract a set of topics related to the textual artifacts analyzed, and estimate the distribution of these latent topics over these artifacts (e.g., what is the probability that a topic will be related to a document). These extracted topics can be leveraged to support various software maintenance tasks, such as traceability link recovery [2], feature location [3], change impact analysis [4], and others.

LDA is established on a solid theoretical foundation, and produces topic models based on the following input parameters: (i) number of topics, the hyper-parameters (ii) α and (iii) β, which influence the distribution of topics per document, and the distribution of the terms per topic, respectively and (iv) the number of iterations needed for the model to converge. In theory, if these parameters are set correctly, LDA would generate a high quality topic distribution which would accurately reflect the underlying topic distribution from the actual data. However, if these parameters are not configured properly, they may lead LDA to produce sub-optimal results.

LDA was originally applied on natural language documents (e.g., books), and was later applied on software artifacts to support SE tasks. The underlying assumption was that software artifacts share the same textual characteristics as text from natural language. This resulted in configuring LDA for software maintenance tasks using the same parameters used for configuring natural language documents. Hence, even though LDA was producing good results on natural language text, it did not always produce the expected results for SE tasks, such as traceability link recovery [5]. A recent study showed that the text extracted from software systems is much more predictable and repetitive as compared to the text from natural language documents [6]. These finding could explain in part why LDA is performing well on natural language and is performing sub-optimally when applied to SE tasks.

In our previous work, we addressed the issue and we proposed LDA-GA [7], an approach that uses Genetic Algorithms (GA) to identify a near-optimal configuration for LDA's parameters. More specifically, LDA-GA uses the coherence of the documents pertaining to the same topic to drive the evolution of the GA, which narrows down the search space of possible configuration of LDA parameters, until a configuration that is near-optimal is found. Our LDA-GA approach was validated on multiple datasets on three SE techniques, namely traceability link recovery, feature location and software artifact labeling, and was shown to produce near-optimal results [7].

In this paper, we instantiate our LDA-GA techniques as a TraceLab experiment that supports traceability link recovery. In addition, we provide the source code (e.g., experiments, components) and datasets used to evaluate LDA-GA, and we provide all the information required by a practitioner to use LDA-GA on other datasets and other SE tasks that require calibration of their LDA techniques, such as feature location, impact analysis, and others. Our data can be access from our online appendix: http://www.cs.wm.edu/semeru/data/tefse13/

This paper addresses the *configurable* Grand Challenge[1] of software traceability. More specifically, it provides a technique that can be used to configure the parameters of the topic model LDA in order to generate the most suitable topics from a corpus of software artifacts, which can be used in IR-based traceability link recovery techniques.

II. Background

This section encapsulates all the information required by a practitioner to use LDA-GA and provides a high-level description of the concepts presented in this paper, namely LDA [1], LDA-GA [7] and TraceLab [8, 9, 10]. For a more formal description of these concepts we refer the interested reader to their original papers.

A. LDA

LDA [1] is a topic model that generates a topic distribution probability for each document analyzed. From an intuitive and simplified point of view LDA addresses the following questions: given a set of document, (i) which are the topics that describe these documents? and (ii) for each document, what is the probability that that document will belong to a particular topic? For SE, the collection of documents (i.e., the corpus) could consist of use cases, source code classes or methods, bug reports or any other textual software artifact.

The corpus of documents is transformed to an $m \times n$ term-by-document (or co-occurrence) matrix M, with m number of unique terms in all the documents, and n representing the number of documents. Each element of M represents the frequency (i.e., the number of times) that a term (from the row) appears in a document (from the column). The term-by-document matrix is used as an input for LDA, which in turn will generate a $k \times n$ topic-by-document matrix θ, with k and n representing the number of topics and number of documents respectively. Each element θ_{ij} represents the probability that the j^{th} document pertains to the i^{th} topic. The fact that k is smaller than n permits LDA to group documents with the same preponderant topics in the same cluster. The coherence of these clusters will be evaluated by our LDA-GA technique (see Section II.B.1) to determine the quality of the LDA model.

For generating LDA models for our approach, we chose the fast collapsed Gibbs sampling over the standard LDA implementation, because it produces equivalent results, in a shorter amount of time [11]. The Gibbs sampling generative model requires the following parameters:

- the number of topics (k) that should be extracted from the corpus; intuitively, this can be considered as the number of clusters required by a clustering algorithm to group the data.
- the number of iterations (n), which represents the number of rounds the Gibbs sampling is applied over the entire corpus. The value should be large enough to allow the LDA model to converge.
- α, the hyper-parameter that influences the smoothing of the topic proportions in documents. Intuitively, larger values of alpha would generate topics that are distributed more uniformly in each document.
- β, the hyper-parameter that influences the smoothing of the terms distribution per topics. Intuitively, larger values of beta would generate results where terms are distributed more uniformly in each topic.

B. LDA-GA

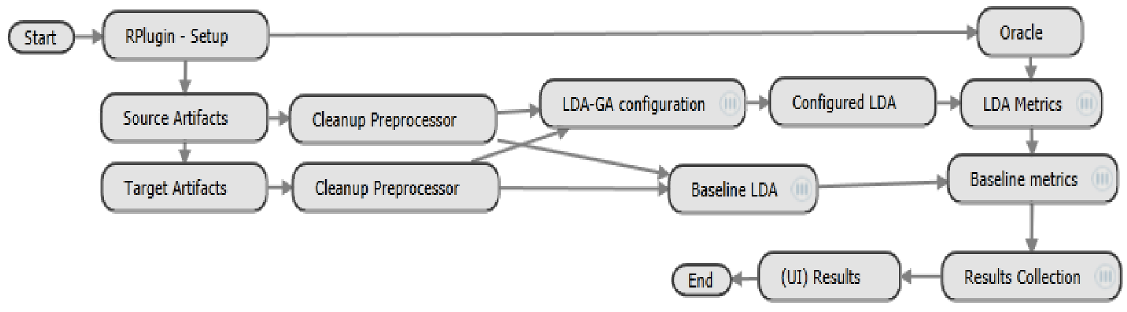
LDA has been used in numerous SE tasks (e.g., traceability link recovery [2, 12], feature location [13], source code labeling [14], etc.) in the same way that it is commonly used in the IR community, which is based on the underlying assumption that source code text and natural language text are the same. However, in light of a recent study that showed that source code is exhibiting different characteristics that natural language text (e.g., it is more predictable and more repetitive) [6], we argue that using the same parameter values used in the IR community may not produce optimal results for SE. Although there were some heuristics [15, 16] for configuring LDA parameters, these approaches focus only on configuring the number of topics, excluding the other hyper-parameters.

LDA-GA [7] addresses this issue by configuring all the LDA parameters in the context of a specific dataset. In other words, given the unique size and characteristics of a dataset, LDA-GA will find an LDA configuration that best fits the particular dataset and is independent of the SE task it will be evaluated in. Our underlying assumption is that the quality of the LDA model (represented by the quality of the clusters produced by the model) should be task independent, and should not be trained using the answer-set of the task. For example, training an LDA model using the answer-set for a traceability link recovery technique would require a priori knowledge of the links, which would defeat the purpose of using an automatic technique to recover them.

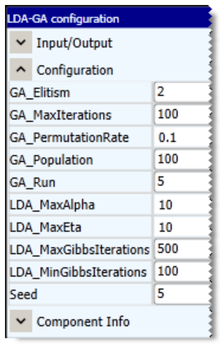
1) Evaluating the LDA configuration. For a given configuration of LDA parameters, an LDA model is computed and evaluated as follows.

First, each document from θ is associated with its dominant topic (i.e., the topic with the highest probability among all the document's topics). Second, the dominant topic of a document represents the cluster the document will belong to. Third, after associating each document with a cluster, two internal quality metrics, namely cohesion and separation, are used to determine how closely related and how dissimilar the clusters are, respectively. These two quality metrics are represented by one value called the Silhouette coefficient [17], which takes into account the maximum distances between documents in the same clusters and the minimum distances between documents pertaining to different clusters. Fourth, the quality of the cluster is interpreted based on the scalar values of the Silhouette coefficient as follows: if the value is close to 1, the clusters are coherent and well separated, and vice-versa for values close to -1. Lastly, the Silhouette coefficient is used as a fitness function (see Section II.B.2) for the GA.

[1] http://www.coest.org/index.php/research-directions/grand-traceability-challenges



(a)


LDA-GA configuration

Input/Output

Configuration

GA_Elitism	2
GA_MaxIterations	100
GA_PermutationRate	0.1
GA_Population	100
GA_Run	5
LDA_MaxAlpha	10
LDA_MaxEta	10
LDA_MaxGibbsIterations	500
LDA_MinGibbsIterations	100
Seed	5

Component Info

(b)

FIGURE 1. (A) THE LDA-GA TRACELAB EXPERIMENT THAT REPLICATES THE CASE STUDY BY PANICHELLA ET AL. [7] ON THE EASYCLINIC SYSTEM; (B) CONFIGURATION PANEL FOR SETTING UP THE GA AND LDA PARAMETERS FOR RUNNING LDA-GA (FOR SPACE LIMITATIONS, THE IMAGE WAS RIGHT-CLIPPED)

2) Finding a (near) optimal LDA Parameter Configuration. The GA [18] that drives LDA-GA to find the near-optimal LDA configuration in the search space of configurations can be summarized as follows. First, a *population* of LDA parameter configurations is generated. Second, each individual (or *chromosome*) of this population will generate an LDA model that will be used to cluster the documents (see Section II.B.1). Third, the clusters for each chromosome will be evaluated using the Silhouette coefficient, which will serve as the *fitness* function (i.e., the function that estimates how suitable an individual is as a candidate for future *generations*). Fourth, the fitness function is the main force to drive the evolution of the genetic algorithm for the next generation (i.e., new population). The new population evolves from the previous population, by applying the following criteria on the individuals of the previous population: (i) preserving the individuals with the highest fitness function (i.e., *elitism*), (ii) applying the *selection operation* (i.e., choose the most suited individuals, with the highest fitness functions to be reproduced), (iii) applying the *crossover operator* (i.e., combine two individuals from the previous population based on some probability) and (iv) applying the *mutation operator* (i.e., the values of the individuals from previous population will be slightly modified based on a probability).

The configuration of the GA used in our LDA-GA approach is a simple GA [18] with: elitism of two individuals, the Roulette wheel selection as the selection operator, the arithmetic crossover for the crossover operator, and the uniform mutation as the mutation operator.

C. TraceLab

TraceLab [8, 9, 10] is a framework designed for creating, executing, and sharing experiments in SE research. TraceLab is developed at DePaul University in collaboration with Kent State University, University of Kentucky, and the College of William and Mary. TraceLab is funded by the National Science Foundation and was initially designed to support traceability link recovery. However, due to its design qualities and characteristics it is well suited to be adapted to other SE tasks, such as feature location, as described in Section III.D. TraceLab was originally designed for the Windows platform, using the Microsoft .NET framework, but has been ported to the Linux and Mac platforms. TraceLab has already been successfully used to implement and reproduce experiments for several research publications [19, 20, 21].

An experiment in TraceLab (see Figure 1(a)) is a directed graph of nodes, where each node represents a tool or technique referred to as a *component* (e.g., *RPlugin-Setup*, *Configured LDA*, etc.). These components can be dragged-and-dropped from the library of components into the graph to add the necessary functionality to an experiment. Edges between nodes indicate both (i) precedence, meaning that all parent components must run before the current component can run, and (ii) data flow, as is often the case in situations where the output of one component is used as the input to the next component (e.g., component *Cleanup Preprocessor* uses as input the output of its parent component *Source Artifacts*). The children of nodes with multiple outgoing edges will be run in parallel, allowing TraceLab to multitask. Every TraceLab experiment has a *Start* and an *End* node (see Figure 1(a)), representing the entry and exit points of the graph. Once the end node is executed, the experiment is complete.

TraceLab ships with the *Component Library*, a comprehensive set of tools and techniques designed for SE research. These can range from simple components, such as data importers, to state-of-the-art components, such as IR techniques supporting traceability links recovery among other techniques. Additionally, TraceLab comes with a software development kit (SDK) for creating custom components, allowing greater flexibility by integrating user techniques into the framework. We refer the user to the TraceLab wiki[2] for more information about using and building components.

III. INSTANTIATING LDA-GA IN TRACELAB

This section details the process of implementing LDA-GA as a TraceLab experiment, and offers some guidelines on adapting LDA-GA to other datasets or SE techniques (Sections III.C and III.D).

A. TraceLab-based LDA-GA Implementation

Figure 1(a) shows the LDA-GA TraceLab experiment that compares two traceability link recovery techniques: one that uses LDA-GA and one that uses an ad-hoc configuration of LDA [12], which is used as a baseline. This experiment is a replication of the case study from [7]. Upon starting the

[2] http://coest.org/coest-projects/projects/tracelab/wiki

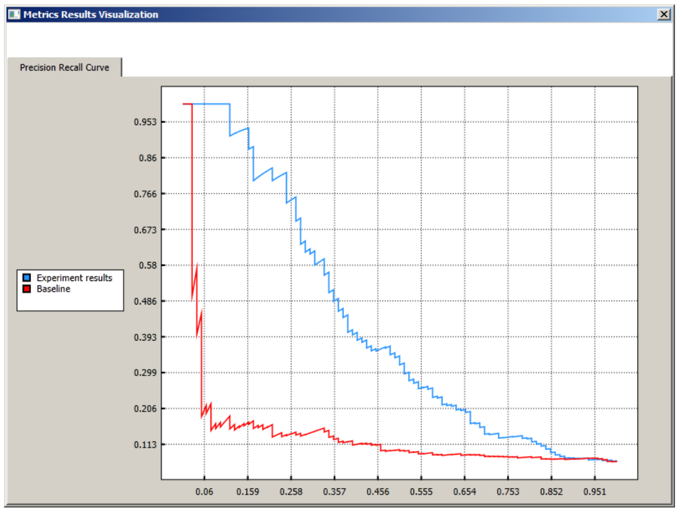


FIGURE 2 PRECISION (Y-AXIS) AND RECALL (X-AXIS) GRAPH FOR EASYCLINIC. THE BASELINE CURVE (RED) IS ON THE BOTTOM AND THE LDA-GA RESULTS CURVE (BLUE) IS ON TOP

experiment, the *RPlugin-Setup* component performs some setup tasks to prepare the components based on R[3] to run in TraceLab. Next, the traceability dataset, composed of the *source* artifacts (e.g., use case documents) and *target* artifacts (e.g., code classes), is loaded into TraceLab and some standard preprocessing is performed (i.e., removing non-alphanumeric symbols, splitting identifiers, removing stop words and stemming). Next, the preprocessed source and target artifacts are sent as inputs to the *LDA-GA Configuration* component. This component can be configured by the user (see Figure 1(b)) with the parameters for the GA (see Section II.B.2) and the parameters of LDA used as a search space for the GA (see Section II.A). Furthermore, the *LDA-GA Configuration* component is responsible for running the LDA-GA algorithm that finds the near-optimal configuration for the LDA parameters. This configuration of parameters is used to generate the LDA model (see *Configured LDA* component) for the traceability recovery task.

Concurrently, the *Baseline LDA* component computes an LDA model (i.e., the baseline) based on an ad-hoc parameter configuration which was used in a previous approach [12]. Both LDA components (i.e., our LDA-GA-based traceability technique and the baseline) output a list of candidate links ranked by their similarity score. Note that the oracle is not used until this point, which emphasized the fact that LDA-GA is a dataset-specific and task-independent approach. The previously established answer set (i.e., oracle) is loaded into TraceLab and fed into the *LDA Metrics* and *Baseline Metrics* components which compute the precision and recall metrics of these two traceability link recovery techniques. The precision-recall curve (see Figure 2) is displayed using the *(UI) Results* component. The results indicate that our LDA-GA-based traceability technique produces better precision and recall results (i.e., the curve starts with 100% precision at 10% recall, and slowly declines to ~80% at 20% recall, ~64% precision at 30% recall, etc.) than a baseline traceability technique that uses an ad-hoc configuration (i.e., the curve suddenly drops to 20% precision at 6% recall).

This result is consistent with the evaluation performed by Panichella et al. [7] (see Figure 3 (a) in [7]). Note that due to the random nature of the GA, the precision and recall curve is not exactly the same as the one produced in Panichella et al.'s experiment [7], but the difference is negligible. For more details about the variability of the results we refer the interested reader to [7].

1) Reusing existing components. Since TraceLab ships with a suite of practical components, we were able to reuse the following components without modification: artifacts importers (for source, target, oracle), corpus preprocessor, metrics computation, and GUI results visualization. In addition, we were able to reuse the RPlugin-Setup component, which allows TraceLab to run any scripts in R. This component is part of the RPlugin library, a separate project that is released along with the Component Library.

2) Implementing new components Our *RPlugin* project uses the LDA implementation from R's *topicmodels* package. We implemented the component responsible for the LDA-GA approach (see *LDA-GA Configuration* component in Figure 1(a)) using an R script which was wrapped and accessed through our *RPlugin* library. The script uses the R package *tm* to transform a corpus into a document-by-term matrix, which is then weighted using the tf-idf scheme. The weighted matrix is used as input for the genetic algorithm found in the R package *GA*, which utilizes the fitness function based on the Silhouette coefficient to generate a near-optimal configuration for LDA's parameters. The LDA parameters, which are the result of the R script, are sent back to the TraceLab workspace.

Note that the LDA-GA component provides the user with the flexibility of changing the configuration options for the GA algorithm (see Section II.B.2) and the search space parameters of LDA used by the GA (see Section II.A) via the component's configuration panel (see Figure 1(b)). More specifically, the configurable GA parameters include: the population size, elitism size, permutation rate for the mutation operation, and maximum number of generations. On the other hand, the LDA-GA parameters for LDA that indicate the search space of possible values for the GA are: the maximum values for α and β (minimum is 0), and the minimum and maximum number of LDA Gibbs iterations. Note that the max number of topics is currently capped by the number of documents in the corpus.

We also implemented a component that takes as input LDA parameters and computes the corresponding LDA model. This component was instantiated twice: once as a component which takes as input a configuration produced by another component (i.e., see Figure 1(a), *Configured LDA*) and once as a component manually configured by a user using ad-hoc parameters (see *Baseline LDA* in Figure 1(a)).

B. Description of the Datasets

We evaluated the two traceability techniques (e.g., LDA-GA and baseline) on the EasyClinic dataset. EasyClinic is a system designed to manage a doctor's office and consists of 20

[3] http://www.r-project.org/

KLOC, 30 use cases (i.e., source artifacts), 47 source code classes (i.e., target artifacts), and 93 correct traceability links between the source and target artifacts.

TraceLab supports a few different corpora formats, and provides the possibility to create custom data importers. For this experiment, we used the *Artifacts Directory Importer* format, in which the corpus is represented by a directory of files. Each file represents an artifact, where the filename is the ID of the artifact and the contents of the file represents the textual description of the artifact. The artifacts are then converted into TraceLab's native corpus format (*TLArtifactsCollection*).

The pre-computed answer sets were imported using the standard *Answer Set Importer* component. The format of the answer set is a tab-delimited file, where the first entry on each line is the ID of a source artifact, and the remaining entries on that line are the IDs of true links of the target artifacts. The imported answer set is then converted into TraceLab's *TLSimilarityMatrix* data type, with all scores set to 1.

C. Adapting LDA-GA to other Datasets

TraceLab was designed to facilitate running existing experiments on different datasets. Therefore, running LDA-GA on another dataset is a straightforward process, which requires adding the path of the other datasets in configuration info pane of the components responsible for importing.

D. Adapting LDA-GA to other SE Techniques

As a general rule of thumb, a user can reuse as many existing components as possible when creating new experiments. A large number of these components are already found in the *Component Library*, while other can be implemented by the user, using the guidelines described in the TraceLab wiki. For example, using LDA-GA for a feature location technique that uses LDA [13] would require replacing the traceability link related components with existing feature location components from the library (e.g., importers, metrics to compute the effectiveness measure, GUI components for generating box plots). As mentioned before, LDA-GA will still run on the dataset and will find the configuration the produces the most coherent topics, while remaining agnostic at the SE task at hand.

Acknowledgment

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

References

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

[2] H. Asuncion, A. Asuncion, and R. Taylor, "Software Traceability with Topic Modeling," in *ICSE*, 2010, pp. 95-104

[3] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature Location in Source Code: A Taxonomy and Survey," *JSEP,* pp. to appear, doi: 10.1002/smr.567, 2012.

[4] M. Gethers, B. Dit, H. Kagdi, and D. Poshyvanyk, "Integrated Impact Analysis for Managing Software Changes," in *ICSE*, 2012, pp. 430-440.

[5] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery," in *ICPC*, 2010, pp. 68-71.

[6] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. T. Devanbu, "On the Naturalness of Software," in *ICSE*, 2012, pp. 837-847.

[7] 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 *ICSE*, 2013, pp. to appear.

[8] J. Cleland-Huang, A. Czauderna, A. Dekhtyar, G. O., J. Huffman Hayes, E. Keenan, G. Leach, J. Maletic, D. Poshyvanyk, Y. Shin, A. Zisman, G. Antoniol, B. Berenbach, A. Egyed, and P. Maeder, "Grand Challenges, Benchmarks, and TraceLab: Developing Infrastructure for the Software Traceability Research Community," in *TEFSE*, 2011, pp. 17-23.

[9] E. Keenan, A. Czauderna, G. Leach, J. Cleland-Huang, Y. Shin, E. Moritz, M. Gethers, D. Poshyvanyk, J. Maletic, J. H. Hayes, A. Dekhtyar, D. Manukian, S. Hussein, and D. Hearn, "TraceLab: An Experimental Workbench for Equipping Researchers to Innovate, Synthesize, and Comparatively Evaluate Traceability Solutions," in *ICSE*, 2012, pp. 1375-1378.

[10] J. Cleland-Huang, Y. Shin, E. Keenan, A. Czauderna, G. Leach, E. Moritz, M. Gethers, D. Poshyvanyk, J. H. Hayes, and W. Li, "Toward Actionable, Broadly Accessible Contests in Software Engineering," in *ICSE,* 2012, pp. 1329-1332.

[11] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling, "Fast Collapsed Gibbs Sampling For Latent Dirichlet Allocation," in *KDD*, 2008, pp. 569-577.

[12] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "On Integrating Orthogonal Information Retrieval Methods to Improve Traceability Link Recovery," in *ICSM*, 2011, pp. 133-142.

[13] L. R. Biggers, C. Bocovich, R. Capshaw, B. P. Eddy, L. H. Etzkorn, and N. A. Kraft, "Configuring Latent Dirichlet Allocation based Feature Location," *ESE,* pp. to appear, doi: 10.1007/s10664-012-9224-x , 2012.

[14] 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 *ICPC*, 2012, pp. 193-202.

[15] S. Grant and J. R. Cordy, "Estimating the Optimal Number of Latent Concepts in Source Code Analysis," in *SCAM*, 2010, pp. 65-74.

[16] T. Griffiths and M. Steyvers, "Finding scientific topics," *National Academy of Sciences*, 2004.

[17] J. Kogan, *Introduction to Clustering Large and High-Dimensional Data*: Cambridge University Press, 2006.

[18] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1989.

[19] A. Panichella, C. McMillan, E. Moritz, D. Palmieri, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "Using Structural Information and User Feedback to Improve IR-based Traceability Recovery," in *CSMR*, 2013, pp. to appear.

[20] B. Dit, E. Moritz, and D. Poshyvanyk, "A TraceLab-based Solution for Creating, Conducting, and Sharing Feature Location Experiments," in *ICPC*, 2012, pp. 203-208.

[21] A. Czauderna, M. Gibiec, G. Leach, Y. Li, Y. Shin, E. Keenan, and J. Cleland-Huang, "Traceability challenge 2011: Using tracelab to evaluate the impact of local versus global idf on trace retrieval," in *TEFSE'11*, Honolulu, HI, USA, 2011.