A Systematic Comparison of Search Algorithms for Topic Modelling—A Study on Duplicate Bug Report Identification

Panichella, Annibale

DOI
10.1007/978-3-030-27455-9_2

Publication date
2019

Document Version
Accepted author manuscript

Published in
Search-Based Software Engineering - 11th International Symposium, SSBSE 2019, Proceedings

Systematic Comparison of Search Algorithms for Topic Modelling - A Study on Duplicate Bug Report Identification

Panichella, Annibale

Publication date
2019

Published in
11th Symposium on Search-Based Software Engineering

Citation (APA)
Panichella, A. (2019). Systematic Comparison of Search Algorithms for Topic Modelling - A Study on Duplicate Bug Report Identification. In 11th Symposium on Search-Based Software Engineering Spinger.

A Systematic Comparison of Search Algorithms for Topic Modelling — A Study on Duplicate Bug Report Identification

Annibale Panichella

Delft University of Technology, The Netherlands
a.panichella@tudelft.nl

Abstract. Latent Dirichlet Allocation (LDA) has been used to support many software engineering tasks. Previous studies showed that default settings lead to sub-optimal topic modeling with a dramatic impact on the performance of such approaches in terms of precision and recall. For this reason, researchers used search algorithms (e.g., genetic algorithms) to automatically configure topic models in an unsupervised fashion. While previous work showed the ability of individual search algorithms in finding near-optimal configurations, it is not clear to what extent the choice of the meta-heuristic matters for SE tasks. In this paper, we present a systematic comparison of five different meta-heuristics to configure LDA in the context of duplicate bug reports identification. The results show that (1) no master algorithm outperforms the others for all software projects, (2) random search and PSO are the least effective meta-heuristics. Finally, the running time strongly depends on the computational complexity of LDA while the internal complexity of the search algorithms plays a negligible role.

Keywords: Topic modeling · Latent Dirichlet Allocation · Search-based Software Engineering · Evolutionary Algorithms · Duplicate Bug Report

1 Introduction

Topic model techniques have been widely used in software engineering (SE) literature to extract textual information from software artifacts. Textual information is often used support software engineers to semi-automated various tasks, such as traceability link retrieval [2], identify bug report duplicates [27], automated summary generator [34, 30], source code labeling [11], and bug localization [22]. Latent Dirichlet Allocation (LDA) is a topic model techniques, which has received much attention in the SE literature due to its ability to extract topics (cluster or relevant words) from software documents. LDA needs to set a number of hyper-parameters. For instance, the Gibbs sampling generative model requires to choose the number of topics K, the number of iteration N, and two hyper-parameters α and β affecting the topic distributions across documents and terms. However, there are no optimal hyper-parameter values that produce "good" LDA models for any dataset. In fact, a prior study showed that untuned

LDA can lead to suboptimal performance and can achieve lower accuracy than simple heuristics based on identifier analysis [11, 12].

To address the tuning challenge, researchers have proposed different strategies over the years [17, 35, 16, 28, 1]. While early attempts focused on the number of topics K as the only parameter to tune, Panichella et al. [28] proposed a search-based approach to tune the LDA hyper-parameters. More specifically, the external performance (e.g., the accuracy) of LDA with a given configuration $[K, N, \alpha, \beta]$ can be indirectly estimated looking at internal cluster quality metrics. In their study, the author used the silhouette coefficient as the driving metric (i.e., the fitness function) to guide genetic algorithms towards finding (near) optimal LDA configurations automatically. Their empirical study showed that LDA settings found with GA dramatically improve the performance of LDA, outperform "off-the-shelf" setting used in previous studies.

Based on the results in [28], Agrawal et al. [1] further investigated search algorithms for tuning LDA. They used Differential Evolution (DE) as alternative meta-heuristic and showed through an extensive study that it often achieves more stable LDA configurations, leading to better topic models than GAs. Besides, they provided further evidence about the usefulness of search-based topic models over "off-the-shelf" LDA settings. Among other results, Agrawal et al. [1] advocated the use of DE as superior meta-heuristics for tuning LDA.

In this paper, we aim to investigate further and compare the performances of multiple meta-heuristics (not only GA and DE) to understand whether there is one meta-heuristic (the "master" algorithm) that constantly dominates all the others. To this aim, we consider the case of *duplicate bug report identification*, which has been often addressed with topic modeling. Duplicate reports are bug reports that describe the same issues but that are submitted by different users to bug tracking systems. Duplicate reports lead to a considerable extra overhead for developers who are in charge of checking and solving the reported issues [20].

We selected seven Java projects from the `Bench4BL` datasets and compared five different meta-heuristics, namely DE, GA, Particle Swarm Optimization (PSO), Simulated Annealing (SA) and Random Search (Ran). Our results show that there is no "master" (dominant) algorithm in search-based topic modeling, although Ran and PSO are significantly less effective than the other meta-heuristics. Besides, DE does not outperforms GA (in terms of both accuracy and running time) when the three meta-heuristics use the same number of fitness evaluations and the stability of LDA is improved using *restarting strategies*.

2 Background and Related work

Document Pre-processing. Applying IR methods requires to perform a sequence of pre-processing steps aimed to extract relevant words from software artifacts (bug reports in our case). The first step is the *term extraction*, in which non-relevant characters (e.g., special characters and numbers) are removed, and compound identifiers are split (e.g., camel-case splitting) [14]. In the second step, a *stop-word list* is used to remove terms/words that do not contribute to the conceptual context of a given artifact, such as prepositions, articles, auxiliary verbs, adverbs, and language keywords. Besides, the *stop-word function* removes words that are shorter than a given threshold (e.g., words with less than three characters). In the last steps, a *stemming* algorithm (e.g., Porter stemmer for English) transform words into their root forms (e.g., verb conjugations). The resulting pre-processed documents are then converted into a *term-by-document matrix* (M). The rows of the matrix denote the terms in the vocabulary after pre-processing (m terms) while the columns denote the documents/artifacts in the corpora (n documents). A generic $M(i,j)$ denotes the weight of the i-th term in the j-th document [3]. The basic weight of each term corresponds to its frequency in a given document (tf = term frequency). However, prior studies suggested using tf-idf (terms frequency with inverse document frequency) which gives lower weights (relevance) to words that appear in most of the documents [5]. The *term-by-document matrix* is then used as input for an algebraic (e.g., Vector Space Model) or probabilistic model (PLSI) to compute the textual similarities among the documents. Such similarities are used differently depending on the SE task to solve. For example, similarities are used to detect duplicated reports with the idea that similar bug reports likely discuss the same bug/issue.

In this paper, we use the following pre-processing steps suggested in the literature [3, 5, 10]: (1) punctuation characters and numbers are removed; (2) splitting compound identifiers with camel-case and snake-case regular expression; (3) a stop-word list for English Language and Java code; (4) stop-word function with a threshold of two characters; (5) words are transformed into their root forms using the Porter stemmer; (6) tf-idf as the weighting schema.

Identifying Duplicate Bug Report. The term-by-document matrix (or its low-dimensional approximation produced by LDA) is then used to compute the Euclidean distance for each pair of documents (bug reports in our case) and compute the ranked list of duplicate bug reports. More specifically, each bug report is used as a query to retrieve the corresponding duplicated reports. The candidate list for each query is therefore determined using the Euclidean distance and sorting the documents in ascending order of distances. Effective IR-methods or topic model should assign better rankings to duplicate reports over non-duplicates. For example, Nguyen et al. [27] combined information retrieval and topic models to detect duplicate reports in an automated fashion. Hindle et al. [20] showed that continuously querying bug reports helps developers to discover duplicates at the time of submitting new bug reports.

Topic modeling with LDA. Latent Dirichlet Allocation (LDA) [8] is a generative probabilistic model for a collection of textual documents (corpora). More specifically, it is a three-level hierarchical Bayesian model which associates documents with multiple topics [8]. In LDA, a topic is a cluster of relevant words in the corpora. Therefore, documents correspond to finite mixtures over a set of K topics. The input of LDA is the term-by-document ($m \times n$) matrix generated using the pre-processing steps described above. LDA generates two distributions of probabilities, one associated with the documents and the other one related the terms in the corpora. The first distribution is the *topic-by-document matrix* (Θ):

a $K \times n$ matrix, where K is the number of topics, n is the number of documents, and the generic entry $\Theta(i,j)$ denotes the probability of the j^{th} document to be relevant to the i^{th} topic. The second distribution is the *word-by-topic matrix* (Φ): an $m \times K$ matrix, where m is the number of words in the corpora, K is the number of topics, and the generic entry $\Phi(i,j)$ denotes the probability of the i^{th} word to belong to the j^{th} topic.

LDA can also be viewed as a dimensional reduction techniquesif the number of topics K is lower than the number of words m in the corpora. Indeed, the term-by-document matrix is decomposed using LDA as follows:

$$\underset{m\times n}{M} \approx \underset{m\times K}{\Phi} \times \underset{K\times n}{\Theta} \tag{1}$$

where K is typically smaller than m. Using Θ, documents can be clustered based on the topics they share based on the corresponding topic probabilities. Documents associated with different topics belong to different topic clusters. Vice versa, documents sharing the same topics belong to the same cluster.

There exist multiple mathematical methods to infer LDA for a given corpora. `VEM` is the applies a deterministic variational EM method using expectation maximization [25]. The fast collapsed `Gibbs sampling` generative model is an iterative process that applied a Markov Chain Monte Carlo algorithm [37]. In this paper, we focus on `Gibbs-sampling` as prior studies showed that it much faster [31], and it can achieve more stable results [17] and better convergence towards the global optimum than VEM [1] in SE documents.

There are four hyper-parameters to set when using the Gibbs sampling generative model for LDA [28, 7]:

- the number of topics K to generate from the corpora;
- α influences the distribution of the topics per document. Smaller α values lead to fewer topics per documents.
- β influences the term distribution in each topic. Smaller β values lead to topics with fewer words.
- the number of Gibbs iterations N; this parameter is specific to the Gibbs sampling generative model.

Stability of the generated topics. LDA is a probabilistic model and, as such, it can produce slightly different models (topics and mixtures) when executed multiple times for the same corpora. Furthermore, different document orderings may lead to different topic distributions [1] (ordering effect). Previous studies (e.g., [21, 1]) suggested different strategies to increase LDA stability, including using random seeds and applying multiple Gibb restarts.

The Gibbs sampling generative method is a stochastic method that performs random samples of the corpora. As any random sampler, the Gibbs method generates random sampling using a random number generator and a starting `seed`. An easy way to achieve the same topics and mixtures consists in using the same initial `seed` when running LDA with the same hyper-parameters and for the same corpora. Another well-known strategy to improve the stability of LDA is restarting the Gibbs sampling to avoid converging toward local optima. For example, Hughes et al. [21] proposed a sparsity-promoting restart and observed dramatic gains due to the restarting. Binkley et al. [6] ran run the Gibbs sampler multiple times suggesting that it reduces the probability of getting stuck in local optima. Recently, Mantyla et al. [24] performed multiple LDA runs and combined the results of different runs through clustering.

In this paper, we use both fixed `seeds` for the sampling and the restarting strategy. More details are provided in Section 3.1.

Automated tuning for LDA. A general problem when using LDA is deciding the hyper-parameters values to adopt when applying it to a specific dataset. Researchers from different communities agree that there is no universal setting that works well for any dataset (e.g., [6, 28, 21]). Different heuristics have been proposed by researchers to find (near) optimal hyper-parameters for a given task [17, 35, 16, 28, 1]. Most of the early approaches focused on the number of topics K to set while using fixed values for α, β and N [17, 16, 35].

Panichella et al. [28] used an internal metric for cluster quality analysis to estimate the fitness of LDA configurations based on the idea that LDA can also be seen as a clustering algorithm. More specifically, they used the silhouette coefficient as the fitness function to guide genetic algorithms, which were used to find LDA hyper-parameters that increased the coefficient values. The *silhouette coefficient* is defined as [28]:

$$s(C) = \frac{1}{n}\sum_{i=1}^{n} s(d_i) \quad \text{with} \quad s(d_i) = \frac{b(d_i) - a(d_i)}{\max\left(a(d_i), b(d_i)\right)} \tag{2}$$

In the equation above, $s(d_i)$ denotes the silhouette coefficient for the document d_i in the corpora; $a(d_i)$ measures the maximum distance of the document d_i to the other documents in the same cluster (cluster cohesion); $b(d_i)$ measures the minimum distance between of the document d_i to another document in a different cluster (cluster separation); $s(C)$ measure the overall silhouette coefficient as the arithmetic mean of the coefficients $s(d_i)$ for all documents in the corpora. $s(C)$ takes values in $[-1, +1]$; larger values indicate better clusters because (on average) the separation is larger than the cohesion of the clusters. While the silhouette coefficient is an internal cluster quality metric, Panichella et al. [28] showed that hyper-parameters that increased the silhouette coefficient also lead to better external performances, such as the accuracy in traceability recovery. Besides, the LDA configurations found with GAs achieve performance that is pretty close to the global optimum. The silhouette coefficient and GA were also used in a later study [29] to configure the whole IR process (including the pre-processing) automatically.

Recently, Agrawal et al. [1] further investigated the challenges of configuring LDA with search algorithms. They showed than Differential Evolution (DE) can generate optimal hyper-parameter values which lead to more stable LDA models (topic and mixtures). Besides, Agrawal et al. also used a different fitness function. An empirical comparison between GA and DE showed that the latter needs fewer generations and produces more stable LDA models than the former. However, in [1] GA and LDA were configured with different termination criteria:

a few dozens of fitness evaluations for DE and thousands of fitness evaluations for GA. Besides, Agrawal et al. [1] did not use standard strategies (e.g., restarting strategies) to produce stable results for both GA and DE. Based on the results in [1], Mantyla et al. [24] used DE in combination with multiple LDA runs to achieve even more stable topics.

While prior studies argued about the superiority of DE over other meta-heuristics for topic modeling, *more research is needed to assess how different meta-heuristics perform when using the same number of fitness evaluations* (e.g., the same termination criteria) *and using random restarting to achieve stable results.* This paper sheds lights on this open question and compares the performance of five different meta-heuristics (not only DE and GA) when configuring LDA for duplicate bug report identification. For the sake of our analysis, we use the silhouette coefficient as the fitness function for all meta-heuristics.

3 Empirical Study

The following research questions steer our study:

- **RQ1**: *Do different meta-heuristics find equally good LDA configurations?* Different meta-heuristics may produce different LDA configurations. Our first research question aims to investigate whether configurations produced by alternative meta-heuristics achieves or not the same accuracy.
- **RQ2**: *Does the running time differ across the experimented meta-heuristics?* Priori study [1] advocated the usage of Differential Evolution (DE) over other meta-heuristics because it requires less running time. With our second research question, we aim to compare the running time of different meta-heuristics when configured with the same number of fitness evaluations.

Benchmark. The benchmark of our study consists of seven datasets from the `Bench4BL` dataset [22] and publicly available in GitHub[1]. The benchmark has been used by Lee at al. to perform a comprehensive reproduction study of state-of-the-art IR-based bug localization techniques. For our study, we selected seven Java project from `Bench4BL`: four projects from the *apache commons library*[2], two projects from Spring[3], and one project from JBoss[4]. The characteristics of the selected projects are reported in Table 1. We chose these seven projects because they have been widely used in the SBSE literature (e.g., [9]) and are well-managed together with issue tracking systems.

For each project, the `Bench4BL` contains (i) issues (from their issue tracking systems) that are explicitly labeled as `bug` by the original developers, and (ii) the corresponding patches/fixes [22]. Each bug report/issue contains (i) the summary (or title), (ii) the description, and (iii) the reporter. Besides, `Bench4BL` also provides the list of duplicated bug reports for each system in the dataset. The

[1] https://github.com/exatoa/Bench4BL
[2] http://www.apache.org/
[3] https://spring.io/
[4] http://www.jboss.org/

Table 1. Characteristics of the projects in our study

System	#Files	#Bug Reports	#Duplicates
Apache commmons collections	525	92	16 (17%)
Apache commons io	227	91	7 (8%)
Apache commons lang	305	217	23 (11%)
Apache commons math	1,617	245	8 (3%)
Spring Datacmns	604	158	15 (9%)
Spring SPR	6,512	130	73 (56%)
JBoss WFly	8,990	984	27 (3%)

percentage of duplicated bug reports ranges between 3% for *apache commons math* and 56% for *Spring SPR*.

Meta-heuristic Selection. We selected the following meta-heuristics:

(1) *Genetic Algorithms* (GAs) have been used in a prior study to configure LDA [28] and the whole IR process [29]. GA is population-based meta-heuristic that evolves a pool of randomly-generated solutions (LDA configurations) through sub-sequent generations. In each generation, solutions are selected based on their fitness values (silhouette coefficient) using the *binary tournament selection*. Fittest solutions (parents) are combined using *binary-simulated crossover* and *gaussian mutation* to form new solutions (offspring). Then, the population for the new generation is formed by selecting the best solutions among parents and offspring (*elitism*).

(2) *Differential Evolution* (DE) is an evolutionary algorithm used by Agrawal et al. [1]. DE is also a population-based meta-heuristic with μ randomly generated solutions. The key difference in DE is that new solutions are generated in each generation by using *differential operators* rather than *genetic operators*. A new solution (LDA configuration) is generated by (1) randomly selecting three solutions a, b, and c from the population; (2) a new solution is generated with the formula: $y_i = a_i + f \times (b_i - c_i)$, where f is the differential weight $\in [0; 2]$; a_i, b_i and c_i denote the i-th elements of the three selected solutions (i.e., the i-th LDA hyper-parameters). The differential operator is applied with a probability $p_c \in [0; 1]$ (crossover probability).

(3) *Particle Swarm Optimization (PSO)* is a population-based meta-heuristic proposed by Eberhart and Kennedy [13]. Similarly to DE and GA, PSO iteratively updates the pool of initial particles (solutions) with initial positions (x), inertia (w), and velocity (v). However, unlike GA and DE that uses crossover (and mutation with GA), PSO updates the solutions toward the best solution in the pool by updating their *positions* and *velocity*.

(4) *Simulated Annealing* (SA) is a meta-heuristic that involves only one solution at a time [36]. One randomly-generated solution x (LDA configuration) is updated through random mutation (neighborhood). If the mutated solution x' improves the fitness function (i.e., $fit(x') < fit(x)$) then SA selects x' as new current solution. If the fitness function decreases with x', the current solution x is still replaced with a probability $\exp^{-\Delta D/T}$, where ΔD is the difference between the cost function for x' and x while T is the temperature. The probability of accepting worst solutions decreases exponentially with ΔD: the higher the difference between the two solutions, the lower the probability of accepting the

worst one. Usually, the parameter T decreases in each iteration to strengthen the exploitation ability of SA.

(5) *Random Search* (Ran) is the simplest search algorithm to implement. It tries K random samples and selects as the final solution (LDA configuration) the one with the best fitness value across all generated trials. Despite its simplicity, random search can outperform more sophisticated meta-heuristics for specific problems [4] and it is often used as a baseline in SSBSE.

Parameter settings. For the search, we opt for the standard parameter setting and search operators suggested in the literature [28, 1]. In particular, for GA we use the following parameter values: population size of 10 LDA configurations; crossover probability p_c=0.9; mutation probability p_m=0.25 (i.e., $1/n$, where n is the number of hyper-parameters for LDA). For DE, we use the following setting: population size μ=10; differential weight factor $f = 0.7$; crossover probability p_c=0.9. SA was configured as follows: neighbors are generated using the Gaussian mutation; the number of steps per temperature ns=10; the number of temperatures nt=5. For PSO, we apply the following setting: population size μ=10; inertia weight w_i=0.9; search weights c_1=c_2=1. The only parameter to set for random search is the number of random solutions to generate.

Termination criteria. To allow a fair comparison, we set all algorithms with the same *stopping criterion*: the search terminates when the maximum number of fitness evaluations (FEs) is reached. Previous studies in search-based topic modeling suggested different values for FEs: Panichella et al. [28] used GA with 100 individuals and 100 generations, corresponding to 10K FEs; Agrawal et al. [1] used DE with 10 individuals and 3 generations, corresponding to 30 FEs. Agrawal et al. [1] argued that fewer FEs are sufficient to achieve good and stable LDA configurations. In addition, too many FEs dramatically impact the overall running time since each LDA execution (individual) is very expensive for large corpora. Based on the motivation by Agrawal et al. [1], we use FEs=50 since it provides a good compromise between performances (TOP$_k$ metrics) and running time in our preliminary experiments. However, we use the same FEs for all meta-heuristics while prior studies [1] used fewer FEs only for DE.

Implementation. For LDA, we use its implementation available in the package `topicmodels` in `R` [18]. We chose this implementation over other implementations (e.g., `Mallet`[5] in Java) because it provides an interface to the original LDA implementation in `C` code by Blei et al. [8]. Furthermore, Binkley et al. [6] showed that the `R` implementation is less sensitive to local optima compared to `Mallet`. The `R` implementation was also used in a prior study concerning LDA configurations for SE tasks [28] and support strategies (e.g., random restarts) to achieve stable LDA models. For the meta-heuristics, we also used their implementation available in `R`: (1) real-coded genetic algorithms from the package `GA` [33]; (2) differential evolution from the package `DEoptim` [26]; (3) random search from the package `randomsearch` [32]; (4) Simulated-Annealing [38], and Particle Swarm Optimization from the package `NMOF` [23].

[5] http://mallet.cs.umass.edu

The R scripts and datasets used in our experiment are publicly available at the following link: `https://apanichella.github.io/tools/ssbse-lda/`.

3.1 Experimental methodology

For each project, we run each meta-heuristic 30 times. In each run, we collected the running time needed to reach the stop condition (see the parameter setting) and the performance metric TOP$_k$. At the end of each run, we use the LDA configuration produced by the meta-heuristic under analysis, and we generated the corresponding LDA model, and the *topic-by-document matrix* (Θ) in particular.

To answer RQ1, we use the TOP$_k$ metric, which measures the performance of an IR-method by checking whether a duplicate bug report to a given query is retrieved within the top k candidate reports in the ranked list. For example, TOP$_5$ is equal to one if the first duplicate report for a given query q is retrieved within the first top $k = 5$ positions in the ranked list. The overall TOP$_k$ metric for a given project is the average of the TOP$_k$ scores achieved for all target reports in the project. More formally, let $|Q|$ be the number of queries (reports) in a given dataset, the TOP$_k$ metric is defined as [20]:

$$TOP_k(Q) = \frac{1}{|Q|} \sum_{q \in Q} in_k(i) \tag{3}$$

where $in_k(i)$ is equal to one if the first duplicated report for the query q is retrieved within the first k positions in the corresponding ranked list. The higher the TOP$_k$, the better the performance of LDA with a given configuration. In this paper, we consider four values of k, i.e., TOP$_5$, TOP$_{10}$, TOP$_{15}$, and TOP$_{20}$.

To answer RQ2, we compare the running time required by the different meta-heuristics to terminate the search process in each independent run. For our analysis, we compare the arithmetic mean for the running time across the 30 independent runs and the corresponding standard deviation.

To assess the statistical significance, we use the Friedman test to compare the performance (TOP$_k$ and running time) of the assessed meta-heuristics over six projects and five different metrics (four TOP$_k$ and the running time). Each meta-heuristic produced 4 (TOP$_k$ metrics) $\times$ 6 (projects) $\times$ 30 (runs) = 720 data points. For statistical analysis, we consider the average (arithmetic mean) of the TOP$_k$ metrics across the 30 runs, resulting in 24 average scores per meta-heuristic. The five distributions (one for each meta-heuristic) are then compared using the Friedman test [15], which is used to assess whether the performance achieved by alternative meta-heuristics significantly differ from one another. Then, to better understand which meta-heuristics performs better, we use the Wilcoxon rank sum test to compare pairs of meta-heuristics. To draw our conclusions, we use the significance level 0.05 for both the Friedman and Wilcoxon tests. Given the large number of pair comparisons with the Wilcoxon tests, we report the number of times (i.e., pair of software projects and TOP$_k$ metrics) a meta-heuristic A performs significantly better than another meta-heuristic B.

Strategies to achieve stable topic modeling. In this paper, we address the stability problem using two standard strategies: seeding and random restart.

When evaluating each LDA configuration (individual), we store both the silhouette coefficient (fitness function) and the random `seed` used to generate the LDA model. Therefore, when the search terminates, LDA is re-run using the best solution (configuration) found across the generation/iterations and using the corresponding random `seed` previously stored. This allows obtaining the same results (silhouette score, topics, and mixtures) even when LDA is re-run multiple times with the same hyper-parameters. Besides, we also used random restarting to improve the stability of the results and reducing the likelihood of reach a local optimum when using the Gibb-sampling method. In particular, the Gibb sampling procedure is restarted $n = 5$ times (independent runs), and the generated topics and mixtures are obtained by averaging the results achieved across the independent results.

4 Empirical Results

Table 2 shows the average (mean) and the standard deviation performance scores (TOP_5, TOP_{10}, TOP_{15}, and TOP_{20}) achieved by the different algorithms in the comparison over 30 independent runs. First, we can notice that there is no "master" (dominant) meta-heuristic that outperforms the others for all software projects. DE, GA, and SA produce the best (largest) TOP_k scores for different projects and with different k values. DE achieves the highest TOP_5 only for two out of seven projects and in only one project for TOP_10. However, in all three cases, DE and GA achieve the same performance score. For all the other projects and metrics, it does not outperform nor compete with other meta-heuristics. Therefore, *our results indicate that DE is not superior to other meta-heuristics* as argued in prior studies.

GA achieves the best scores in 17 cases (six projects with different TOP_k metrics). For the projects `io`, `math`, and `wfly`, GA outperforms all other meta-heuristics according to all TOP_k scores. The differences with the second highest scores range between 2% (`math` with TOP_5) and 21% (`wfly` with TOP_5). It is worth noting that these three projects present the lowest percentages of duplicated bug reports (<=8%) compared to the other projects (see Table 1). These results suggest that GA is likely more effective on projects with very few duplicate bug reports. For the projects `datacmns`, `lang`, and `spr`, GA achieves the best TOP_k scores only for $k = 5$ (for both projects) and $k = 10$ (for `spr`). For larger k values, SA produces the best TOP_k scores among the five meta-heuristics.

In general, SA achieves the best TOP_k scores in 12 cases (four projects with different TOP_k metrics). Independently from the TOP_k metric, SA is the best meta-heuristic for `collections`, which is the smallest projects (<100 bug reports) in our benchmark. The differences with the second best meta-heuristic vary between 4% and 5%. For other three projects, namely `datacmns`, `lang`, and `spr`, SA achieves the best results only for larger values of k.

Random search never produces the best TOP_k scores. However, it does produce better average TOP_k scores than DE and GA for `collections` and `lang`. Finally, PSO produces the lowest TOP_k scores than all other meta-heuristics

Table 2. Mean and standard deviation of the performance scores achieved by the evaluated meta-heuristics

System	Metric	DE		GA		Ran		SA		PSO	
		Mean	S.d.	Mean	S.d.	Mean	S.d.	Mean	S.d.	Mean	S.d.
Collections	TOP_5	0.87	0.09	0.90	0.08	0.90	0.07	**0.95**	0.05	0.80	0.07
	TOP_{10}	0.88	0.10	0.90	0.07	0.91	0.08	**0.95**	0.05	0.84	0.07
	TOP_{15}	0.89	0.10	0.90	0.07	0.91	0.08	**0.96**	0.04	0.84	0.07
	TOP_{20}	0.90	0.08	0.90	0.07	0.91	0.08	**0.96**	0.04	0.85	0.07
Datacmns	TOP_5	0.44	0.10	**0.47**	0.10	0.41	0.12	0.37	0.05	0.28	0.09
	TOP_{10}	0.52	0.12	**0.54**	0.11	0.51	0.12	0.52	0.07	0.39	0.13
	TOP_{15}	0.54	0.13	0.57	0.11	0.53	0.15	**0.61**	0.13	0.42	0.14
	TOP_{20}	0.56	0.13	0.58	0.11	0.55	0.14	**0.63**	0.15	0.46	0.13
IO	TOP_5	0.51	0.12	**0.54**	0.11	0.45	0.16	0.41	0.17	0.22	0.05
	TOP_{10}	0.55	0.12	**0.61**	0.12	0.50	0.18	0.54	0.27	0.36	0.07
	TOP_{15}	0.56	0.11	**0.64**	0.10	0.54	0.15	0.55	0.28	0.44	0.07
	TOP_{20}	0.61	0.12	**0.68**	0.10	0.61	0.15	0.56	0.26	0.52	0.05
Lang	TOP_5	**0.58**	0.11	**0.58**	0.05	0.57	0.05	0.50	0.05	0.38	0.13
	TOP_{10}	0.62	0.12	0.62	0.06	0.64	0.05	**0.68**	0.07	0.45	0.09
	TOP_{15}	0.65	0.11	0.64	0.06	0.67	0.05	**0.69**	0.05	0.48	0.09
	TOP_{20}	0.67	0.12	0.65	0.06	0.69	0.04	**0.71**	0.05	0.49	0.10
Math	TOP_5	0.45	0.09	**0.47**	0.12	0.45	0.14	0.43	0.04	0.38	0.19
	TOP_{10}	0.51	0.11	**0.57**	0.12	0.50	0.16	0.48	0.10	0.41	0.19
	TOP_{15}	0.51	0.10	**0.58**	0.12	0.50	0.16	0.50	0.10	0.42	0.19
	TOP_{20}	0.51	0.10	**0.58**	0.12	0.51	0.15	0.50	0.11	0.44	0.18
Spr	TOP_5	**0.62**	0.04	**0.62**	0.06	0.58	0.06	0.53	0.08	0.53	0.12
	TOP_{10}	**0.65**	0.04	**0.65**	0.05	0.61	0.06	**0.65**	0.03	0.57	0.11
	TOP_{15}	0.67	0.05	0.67	0.05	0.63	0.07	**0.72**	0.09	0.61	0.10
	TOP_{20}	0.69	0.04	0.69	0.04	0.66	0.06	**0.76**	0.10	0.63	0.09
WFly	TOP_5	0.31	0.08	**0.53**	0.10	0.30	0.09	0.44	0.10	0.13	0.03
	TOP_{10}	0.33	0.08	**0.53**	0.09	0.31	0.10	0.48	0.09	0.15	0.03
	TOP_{15}	0.33	0.09	**0.53**	0.09	0.32	0.10	0.50	0.10	0.16	0.02
	TOP_{20}	0.33	0.09	**0.53**	0.09	0.32	0.10	0.51	0.09	0.16	0.02

and for all projects in our study. Therefore, it is not a suitable meta-heuristic for topic models, at least in the context of duplicate bug report identifications.

The differences among the different meta-heuristics are statistically significant according to the Friedman test, whose resulting p-value is $3.79\times\ 10^{-10}$. To better understands which meta-heuristics performs statistically better (or worse) than others, Tables 4-7 report the number of projects in which each meta-heuristic (rows in the tables) significantly outperforms another meta-heuristic (columns in the tables) according to the Wilcoxon test. Instead, Table 8 reports the ranking produces by the Friedman tests. According to the statistical results, GA is ranked first, followed by SA and DE, respectively. Instead, Random search and PSO are the bottom two meta-heuristics. While GA was ranked first, we can notice that it does not significantly outperform all other meta-heuristics for all projects. However, it significantly outperforms Random Search and PSO in most of the projects. It outperforms SA in most of the projects only for POS_5 while for $k > 5$, the two meta-heuristics are comparable. DE (that is ranked third) never outperforms GA according to the Wilcoxon test. Vice versa, GA significantly outperforms DE in three out of seven projects for $POS_{k>5}$.

Table 3. Number of projects in which one meta-heuristic (row) statistically outperforms another one meta-heuristic (column) according to the Wilcoxon test.

Table 4. TOP$_5$

Vs.	DE	GA	Ran	SA	PSO
DE	-	0	3	3	7
GA	1	-	4	5	7
Ran	0	0	-	2	7
SA	2	1	2	-	7
PSO	0	0	0	0	-

Table 5. TOP$_{10}$

Vs.	DE	GA	Ran	SA	PSO
DE	-	0	2	1	7
GA	3	-	4	1	7
Ran	1	0	-	0	6
SA	3	1	3	-	7
PSO	0	0	0	0	-

Table 6. TOP$_{15}$

Vs.	DE	GA	Ran	SA	PSO
DE	-	0	1	1	7
GA	3	-	4	1	7
Ran	0	1	-	0	6
SA	3	2	4	-	7
PSO	0	0	0	0	-

Table 7. TOP$_{20}$

Vs.	DE	GA	Ran	SA	PSO
DE	-	0	1	1	7
GA	3	-	4	2	7
Ran	0	1	-	0	6
SA	3	1	5	-	7
PSO	0	0	0	0	-

Table 8. Ranking produced by the Friedman Tests

Meta-heuristic	Ranking
GA	2.085714
SA	2.457143
DE	2.571429
Random	3.457143
PSO	4.428571

Table 9. Mean and standard deviation of the running time required by the evaluated meta-heuristics to perform 50 fitness evaluations

System	DE		GA		Ran		SA		PSO	
	Mean	S.d.	Mean	S.d.	Mean	S.d.	Mean	S.d.	Mean	S.d.
Collections	14	2.34	11	2.05	7	5.15	6	0.60	14	1.25
Datacmns	91	14.53	72	14.19	46	32.66	72	36.82	85	15.36
Io	15	2.69	11	1.91	9	6.07	9	1.90	15	0.98
Math	118	16.40	96	24.97	66	47.61	138	85.66	129	20.24
Lang	92	8.83	72	11.98	48	34.14	82	58.55	80	14.81
Spr	64	12.02	58	8.00	37	26.35	85	66.07	71	10.28
WFly	6554	62.36	5196	130.78	3653	262.42	5124	501.38	6433	94.56

There is no "master" (dominant) meta-heuristic when configuring topic models for duplicate bug report identification. GA and SA perform better than other meta-heuristics but not consistently across projects. Random search and PSO are the least effective meta-heuristics.

Table 9 reports the mean (and the standard deviation) running time required by the evaluated meta-heuristics to reach the same stopping criterion (50 FEs) across 30 independent runs. As expected, random search is the fastest among all meta-heuristics since it does not involve any solution selection and update (e.g., mutation). For what regards the other meta-heuristics, we can notice that their running does not differ substantially. On average, the difference between each pair of meta-heuristics is lower than 10%, and this small difference is mostly due to the computational complexity of the different individual operators. For example, GA is faster than SA in three projects but slower in three other projects. DE and PSO are instead slightly slower than DE and SA, although the differences are small and in some cases almost negligible (e.g., few additional seconds for the project *collections*). These results contradict what reported by Agrawal et al. [1], who used fewer fitness evaluations with DE and many more with GA. In this study, we use the same number of fitness evaluations for all meta-heuristics to allow a fair comparison. When using the same stopping criterion, DE is slightly slower than GA. This confirms previous results in evolutionary computation (e.g., [19]) that showed how the extra overhead in DE is due to the computation complexity of differential operators. Indeed, a single generation of DE is on average four times more expensive than one single generation with GA [19].

The running time strongly depends on the number of fitness evaluation performed during the search (time to infer LDA). Instead, the internal complexity of the meta-heuristics is small or negligible.

Threats to validity. *Construct validity.* All meta-heuristics are implemented in `R` and were executed with the same stopping criterion. Furthermore, we use *seeding* and *random restarts* for all meta-heuristics to alleviate the instability of the LDA results. *Internal validity.* We drew our conclusions by executing 30 independent runs to address the random natures of the evaluated meta-heuristics. Besides, we use the Wilcoxon and the Friedman tests to assess the statistical significance of the results. We use TOP_k as the performance metric because it is a standard performance metric in duplicate bug report identification. *External validity.* In our study, we consider seven open source projects from the `Bench4BL` dataset [22]. Assessing the different meta-heuristics and selecting more projects is part of our future plan.

5 Conclusion and Future Work

In this paper, we empirically compare different meta-heuristics when applied to tune LDA parameters in an automated fashion. We focus on topic-model based identification of bug report duplicates, which is a typical SE task and addressed in prior studies with topic model and IR methods (e.g., [27, 20]). Experimental results on seven Java projects and their corresponding bug reports show that multiple meta-heuristics are comparable across different projects, although random search and PSO are least effective than other meta-heuristics. Therefore, *no meta-heuristic outperforms all the others* as advocated in prior studies. However, our conclusions hold for the problem of identifying duplicate bug reports. Therefore, different results may be observed in different SE tasks. Our future work will focus on extending our study by (i) comparing more meta-heuristics, (ii) considering more projects and (iii) evaluating other SE tasks.

References

1. Agrawal, A., Fu, W., Menzies, T.: What is wrong with topic modeling? and how to fix it using search-based software engineering. Information and Software Technology **98**, 74–88 (2018)
2. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A.: Information retrieval models for recovering traceability links between code and documentation. In: The 16th IEEE International Conference on Software Maintenance. pp. 40–51 (2000)
3. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley (1999)

4. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research **13**(2), 281–305 (2012)
5. Binkley, D., Lawrie, D.: Information retrieval applications in software maintenance and evolution. Encyclopedia of Software Engineering (2009)
6. Binkley, D., Heinz, D., Lawrie, D., Overfelt, J.: Source code analysis with lda. Journal of Software: Evolution and Process **28**(10), 893–920 (2016)
7. Bird, C., Menzies, T., Zimmermann, T.: The art and science of analyzing software data. Elsevier (2015)
8. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet Allocation. The Journal of Machine Learning Research **3**, 993–1022 (2003)
9. Campos, J., Ge, Y., Albunian, N., Fraser, G., Eler, M., Arcuri, A.: An empirical evaluation of evolutionary algorithms for unit test suite generation. Information and Software Technology **104**, 207–235 (2018)
10. Capobianco, G., De Lucia, A., Oliveto, R., Panichella, A., Panichella, S.: On the role of the nouns in IR-based traceability recovery. In: The 17th IEEE International Conference on Program Comprehension (2009)
11. De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A., Panichella, S.: Using IR methods for labeling source code artifacts: Is it worthwhile? In: The 20th IEEE International Conference on Program Comprehension (ICPC). pp. 193–202 (2012)
12. De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A., Panichella, S.: Labeling source code with information retrieval methods: an empirical study. Empirical Software Engineering **19**(5), 1383–1420 (Oct 2014)
13. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: The 6th Intern. Symposium on Micro Machine and Human Science. pp. 39–43 (1995)
14. Enslen, E., Hill, E., Pollock, L.L., Vijay-Shanker, K.: Mining source code to automatically split identifiers for software analysis. In: The 6th International Working Conference on Mining Software Repositories. pp. 71–80 (2009)
15. García, S., Molina, D., Lozano, M., Herrera, F.: A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 special session on real parameter optimization. Journal of Heuristics **15**(6), 617–644 (Dec 2009)
16. Grant, S., Cordy, J.R.: Estimating the optimal number of latent concepts in source code analysis. In: The 10th International Working Conference on Source Code Analysis and Manipulation. pp. 65–74 (2010)
17. Griffiths, T.L., Steyvers, M.: Finding scientific topics. Proc. of the National Academy of Sciences **101**(Suppl. 1), 5228–5235 (2004)
18. Grün, B., Hornik, K.: topicmodels: An R package for fitting topic models. Journal of Statistical Software **40**(13), 1–30 (2011)
19. Hegerty, B., Hung, C.C., Kasprak, K.: A comparative study on differential evolution and genetic algorithms for some combinatorial problems. In: The 8th Mexican international conference on artificial intelligence. pp. 9–13 (2009)
20. Hindle, A., Onuczko, C.: Preventing duplicate bug reports by continuously querying bug reports. Empirical Software Engineering **24**(2), 902–936 (2019)
21. Hughes, M., Kim, D.I., Sudderth, E.: Reliable and scalable variational inference for the hierarchical dirichlet process. In: Artificial Intelligence and Statistics. pp. 370–378 (2015)
22. Lee, J., Kim, D., Bissyandé, T.F., Jung, W., Le Traon, Y.: Bench4bl: reproducibility study on the performance of ir-based bug localization. In: The 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 61–72. ACM (2018)

23. Manfred Gilli, D.M., Schumann, E.: Numerical Methods and Optimization in Finance (NMOF) (2011)
24. Mantyla, M.V., Claes, M., Farooq, U.: Measuring lda topic stability from clusters of replicated runs. In: The 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. p. 49. ACM (2018)
25. Minka, T., Lafferty, J.: Expectation-propagation for the generative aspect model. In: The 18th conference on Uncertainty in artificial intelligence. pp. 352–359. Morgan Kaufmann Publishers Inc. (2002)
26. Mullen, K., Ardia, D., Gil, D., Windover, D., Cline, J.: Deoptim: An r package for global optimization by differential evolution. Journal of Statistical Software **40**(6), 1–26 (2011)
27. Nguyen, A.T., Nguyen, T.T., Nguyen, T.N., Lo, D., Sun, C.: Duplicate bug report detection with a combination of information retrieval and topic modeling. In: The 27th IEEE/ACM International Conference on Automated Software Engineering. pp. 70–79 (2012)
28. Panichella, A., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D., De Lucia, A.: How to effectively use topic models for software engineering tasks? An approach based on genetic algorithms. In: The International Conference on Software Engineering. pp. 522–531. IEEE Press (2013)
29. Panichella, A., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D., De Lucia, A.: Parameterizing and assembling ir-based solutions for se tasks using genetic algorithms. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). vol. 1, pp. 314–325. IEEE (2016)
30. Panichella, S., Panichella, A., Beller, M., Zaidman, A., Gall, H.C.: The impact of test case summaries on bug fixing performance: An empirical investigation. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). pp. 547–558 (May 2016)
31. Porteous, I., Newman, D., Ihler, A., Asuncion, A., Smyth, P., Welling, M.: Fast collapsed gibbs sampling for latent dirichlet allocation. In: The 14th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 569–577. ACM (2008)
32. Richter, J.: randomsearch: Random Search for Expensive Functions (2019)
33. Scrucca, L.: GA: A package for genetic algorithms in R. Journal of Statistical Software, Articles **53**(4), 1–37 (2013)
34. Sridhara, G., Hill, E., Muppaneni, D., Pollock, L.L., Vijay-Shanker, K.: Towards automatically generating summary comments for java methods. In: The 25th IEEE/ACM International Conference on Automated Software Engineering. pp. 43–52. ACM Press (2010)
35. Teh, Y., Jordan, M., Beal, M., Blei, D.: Hierarchical Dirichlet processes. Journal of the American Statistical Association **101**(476), 1566–1581 (2006)
36. Van Laarhoven, P.J., Aarts, E.H.: Simulated annealing. In: Simulated annealing: Theory and applications, pp. 7–15. Springer (1987)
37. Wei, X., Croft, W.B.: Lda-based document models for ad-hoc retrieval. In: The 29th Annual International Conference on Research and Development in Information Retrieval. pp. 178–185. ACM (2006)
38. Yang Xiang, Gubian, S., Suomela, B., Hoeng, J.: Generalized simulated annealing for efficient global optimization: the GenSA package for R. The R Journal Volume 5/1, June 2013 (2013)