A Systematic Comparison of Search-Based Approaches for LDA Hyperparameter Tuning

Panichella, A.

DOI
10.1016/j.infsof.2020.106411

Publication date
2021

Document Version
Final published version

Published in
Information and Software Technology

Information and Software Technology 130 (2021) 106411

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A Systematic Comparison of search-Based approaches for LDA hyperparameter tuning

Annibale Panichella *

Delft University of Technology, The Netherlands

ARTICLE INFO

Keywords:
Topic modeling
Latent dirichlet allocation
Search-based software engineering
Metaheuristic search
Duplicate bug report
Hyperparameter optimization

ABSTRACT

Context: Latent Dirichlet Allocation (LDA) has been successfully used in the literature to extract topics from software documents and support developers in various software engineering tasks. While LDA has been mostly used with default settings, previous studies showed that default hyperparameter values generate sub-optimal topics from software documents.

Objective: Recent studies applied meta-heuristic search (mostly evolutionary algorithms) to configure LDA in an unsupervised and automated fashion. However, previous work advocated for different meta-heuristics and surrogate metrics to optimize. The objective of this paper is to shed light on the influence of these two factors when tuning LDA for SE tasks.

Method: We empirically evaluated and compared seven state-of-the-art meta-heuristics and three alternative surrogate metrics (i.e., fitness functions) to solve the problem of identifying duplicate bug reports with LDA. The benchmark consists of ten real-world and open-source projects from the `Bench4BL` dataset.

Results: Our results indicate that (1) meta-heuristics are mostly comparable to one another (except for random search and CMA-ES), and (2) the choice of the surrogate metric impacts the quality of the generated topics and the tuning overhead. Furthermore, calibrating LDA helps identify twice as many duplicates than untuned LDA when inspecting the top five past similar reports.

Conclusion: No meta-heuristic and/or fitness function outperforms all the others, as advocated in prior studies. However, we can make recommendations for some combinations of meta-heuristics and fitness functions over others for practical use. Future work should focus on improving the surrogate metrics used to calibrate/tune LDA in an unsupervised fashion.

1. Introduction

Researchers have successfully used textual information in software documents (e.g., bug reports, use cases) to support software engineers performing various tasks, such as traceability link retrieval [1], identify bug report duplicates [2], automated summary generator [3,4], source code labeling [5], and bug localization [6]. Information retrieval (IR) and topic modeling (TM) techniques are key technologies to extract, manage, and analyze textual information in software documents. The intuitive idea is that software documents contain textual information written in natural language (e.g., comments in the source code, source code identifiers) that can be treated with IR and TM methods.

Latent Dirichlet Allocation (LDA) is a topic modeling technique proposed by Blei et al. [7] for textual corpora, and it has received much attention in the SE literature. LDA is a generative, probabilistic model that extracts topics (set of words) that best describes a corpus (e.g., software documents). However, LDA comes with a number of hyper-parameters that one needs to set before use. For instance, the Gibbs sampling generative model requires to choose the number of topics K, the number of iterations N, and two hyperparameters —α and β— affecting the topic distributions across documents and terms. Prior studies showed that untuned LDA leads to sub-optimal performance (e.g., [8,9]). Indeed, simple heuristics based on identifier analysis [5,10] can outperform LDA with default settings. Researchers also showed that there are no optimal hyperparameter values that can be used "off--the-shelf" for any dataset (e.g., [8,9]).

Consequently, researchers have proposed different strategies over the years [8,9,11–13] to automate the LDA tuning process. Early attempts focused on the number of topics K as the main parameter to tune while using a very large number of iterations (e.g., $N > 100$) and

* Corresponding author.
E-mail address: a.panichella@tudelft.nl.

https://doi.org/10.1016/j.infsof.2020.106411
Received 15 January 2020; Received in revised form 6 September 2020; Accepted 9 September 2020
Available online 12 September 2020

"off-the-shelf" values for α and β [14]. The tuning process leverages surrogate metrics (e.g., divergence [14] and coherence [15,16]) that correlate with the accuracy of LDA (e.g., agreement with human-generated topics). Hyperparameter values that optimize surrogate metrics can be produced using search algorithms, such as sampling [15], ant colony [17], and particle swarm optimization [18]. However, these approaches have been proposed and evaluated for traditional textual corpora (e.g., biomedicine [18]). This poses an important question: *Do tuning approaches used for conventional corpora work for software documents as well*?

In the context of SE, Panichella et al. [9] applied meta-heuristic search to tune multiple LDA hyperparameters [*K, N, α, β*] based on internal cluster quality metrics. In their study, the authors used the *silhouette coefficient* as the fitness function to optimize with genetic algorithms (GAs). Their empirical study showed that LDA settings found with GA improve the performance of LDA in three SE tasks, also outperforming "off-the-shelf" settings used in previous studies. Agrawal et al. [8] further investigated search algorithms for tuning LDA. They used Differential Evolution (DE) as an alternative meta-heuristic and the *raw score* as an alternative surrogate metric to optimize. They provided further evidence about the usefulness of search-based topic models over "off-the-shelf" LDA settings. Among other results, Agrawal et al. [8] advocated using DE as a better meta-heuristic for tuning LDA. Besides, they also reported that DE leads to more stable LDA configurations and better topic models than GAs.

In our previous study [19], we empirically assessed five different meta-heuristics to tune LDA and *identifying duplicate bug reports* in seven projects from the `Bench4BL` dataset [6]. Duplicate reports describe the same issues but are submitted by developers to bug tracking systems. Duplicate reports lead to considerable extra overhead for developers who are in charge of checking and solving the reported issues [20]. Given the textual nature of bug reports, duplicates can be identified using topic models, and LDA in particular.

The goal of our previous study was to determine *to what extent the choice of meta-heuristic matter for SE tasks beyond* DE *and* GA. We compared five meta-heuristics to tune LDA using the silhouette coefficient as the target fitness function to optimize. The outcome of the study was that there is no "master" (dominant) algorithm in search-based topic modeling, as advocated in prior studies. However, we identified combinations of search-algorithms and meta-heuristics that are more effective than others. Finally, we found out that random search and particle swarm optimization are significantly less effective (i.e., achieve lower TOP_K scores) than the other meta-heuristics. =

In the current article, we extend our conference paper by conducting a larger study with more meta-heuristics and different surrogate metrics (fitness functions) in the context of *duplicate bug report identification*. With respect to our previous paper, this article makes the following contributions:

- We extend the set of meta-heuristics from five (DE, GA, Random Search, Simulated Annealing, and Particle Swarm Optimization) in the conference paper, to seven meta-heuristics, by adding Covariance Matrix Adaptation Evolution Strategy (CMA-ES) and Stochastic Hill Climbing. These meta-heuristics belong to different classes of search algorithms, i.e., global, local, adaptive, and evolutionary solvers.
- We investigate three surrogate metrics as alternative fitness functions to guide the search. In the conference version, we considered only the silhouette coefficient [9], which was defined for software engineering corpora. In this article, we also consider (1) the *raw score* proposed by Agrawal et al. [8] for SE documents, and (2) the coherence metrics proposed by Mimno et al. [15] for traditional corpora.
- We compare the selected meta-heuristics and surrogate metrics (fitness functions) in terms of both (1) quality of topics generated by the tuned LDA and (2) running time.
- We extended the benchmark used in the empirical evaluation from seven (conference paper) to ten projects selected from the `Bench4BL` corpus [6].
- We increase the number of repetitions (from 25 to 100) to increase the statistical power of our analysis. Further, we strengthen the empirical evaluations with a more sound statistical analysis, which now includes the Friedman test [21], the post-hoc Nemenyi test [22], and the two-way permutation test [23]. The Friedman test determines whether different LDA tuning strategies are statistically different. The post-hoc Nemenyi test ranks the different strategies across multiple datasets and determines for which pairs of strategies the statistical significance holds. Finally, the permutation test allows us to establish whether some co-factors significantly affect the achieved results. Note that the three tests are non-parametric.

The key finding of our study is that no search-based approach (a combination of meta-heuristics and surrogate metrics) systematically outperforms the others in all studied projects. However, we identified the top combinations that produce high-quality topics and with a lower overhead:

- Genetic algorithms with the *silhouette coefficient* are faster than other combinations and produce topics that are as good as those achievable with the alternative approaches. Therefore, we highly recommend this combination for larger projects or when the time for tuning LDA is limited.
- Stochastic hill climbing or simulated annealing with either *silhouette coefficient* or *topic coherence* for small projects or when the running time is not a strong constraint.

Finally, our results confirm the importance of tuning LDA to produce high-quality topics and achieve "good" results in software engineering tasks. For the problem of identifying duplicates reports, practitioners that use LDA with tuned parameters could potentially identify (on average) as twice as many duplicates reports by inspecting the top five most similar past reports. While prior studies [8,9] investigated the benefits of tuning LDA hyperparameters for various SE problems (e.g., traceability link retrieval, feature locations), to the best of our knowledge, this is the first work that systematically compares multiple meta-heuristics and surrogate metrics in the context of SE artifacts.

The remainder of this article is organized as follows. Section 2 describes basic concepts about IR methods and LDA. It also summarizes the main related studies focusing on (i) identifying duplicate bug reports, (ii) tuning LDA, and (iii) addressing topics instability. Section 3 overviews the search-based approaches proposed in the literature to automatically tune LDA for software and generic documents. Section 4 describes the design of our empirical study, while Section 5 analyzes the collected results. Section 6 discusses the benefits of tuning LDA over using "off-the-shelf" configurations. It also provides guidelines for tuning LDA and its relations with regards to tuning search-based approaches. Section 7 discusses the threats to validity. Section 8 concludes the paper.

2. Background and related work

This section introduces basic concepts related to *textual analysis*. It also summarizes the related work regarding the application of *topic modeling* and *textual analysis* methods to identify duplicate bug reports. Finally, it describes LDA and related tuning challenges.

2.1. Textual analysis

Applying textual analysis methods requires to perform three macro-steps: (1) the pre-processing aims to extract relevant information from textual documents, (2) representing the extracted information using a mathematical model, and (3) computing the textual similarity among

documents based on the chosen mathematical model. The next subsections detail these macro-steps.

2.1.1. Pre-processing

Document pre-processing aims to extract relevant words from documents (bug reports in our case), and remove potential noise [24]. Bug reports contain both sentences in natural language and code snippets. Therefore, the first step consists in *normalizing* the code snippets into natural-language-like text. Once the code snippets are normalized, both natural language and code snippets are transformed using the standard pre-processing.

Source-code normalization. The normalization includes several steps that are specific to source code analysis. The first step is removing all programming-language specific punctuation marks, such as brackets (e.g., "`{},[]`"), comment delimiters (e.g., `/**`, `//` in Java), and semicolons. In the second step, we split compound identifiers (e.g., camelcase splitting) into their constituent morphemes [25]. To this aim, we use the `snakecase` library in `R`, which includes multiple case transformation utilities. Finally, we removed programming language keywords (`public`, `class`, `void`, etc.). This filters out words that are too generic and do not contribute to the *textual semantic* of the code.

Standard pre-processing. The first pre-processing step is the *term extraction*, in which non-relevant characters (e.g., special characters and numbers) and spaces are removed. In the second step, a *stop-word list* is used to remove terms/words that do not contribute to the conceptual content of the software documents being pre-processed, such as prepositions, articles, auxiliary verbs, and adverbs.

The *stop-word function* removes words that are shorter than a given threshold (e.g., words with less than two/three characters). In the last step, a *stemming* algorithm (e.g., Porter stemmer for English [26]) transforms words into their root forms. For example, verb conjugations are converted into their infinite forms, plural nouns in their singular forms, and so on.

This paper uses the following pre-processing steps suggested in the literature [27–29]. First, we removed punctuation characters, numbers, and special characters. To remove non-relevant words, we use the stop-word list for the English Language available in the `tm` package in `R`. Furthermore, we enlarge the stop-word list with the 51 reserved words in Java and 55 commonly-used words in Java programs that are too generic (e.g., "set", "get", "args"). We also use a stop-word function that removes all words with fewer than three characters. Finally, we used the Porter stemmer to transform extracted words into their root forms. Notice that these steps are applied to both natural language sentences and normalized code snippets.

2.1.2. Term-by-document matrix

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 entry *M* (*i, j*) denotes the weight of the *i*-th term in the *j*-th document [27]. The most natural weighting scheme is the *term frequency* (*tf*), which counts the number of times (frequency) each term appears in a given document. Prior studies recommend using *tf-idf* (terms frequency with inverse document frequency), which gives lower weights (relevance) to words that appear in most of the documents [28]. The *tf-idf* scheme assigns to each term *t* appearing in a document *d* the weight [24]:

$$tf - idf(t,d) = tf(t,d) \times \log \frac{n}{td_t} \quad (1)$$

In Eqn. 1, *n* is the number of documents in the collections; *tf(t, d)* denotes the frequency of *t* in the document *d*; and td_t measures the number of documents which the term *t* appears in (document frequency) within the collection. In this paper, we used *tf-idf* as the weighting scheme.

2.1.3. Mathematical model

The *term-by-document matrix* is then used as input for an algebraic (e.g., Vector Space Model) or probabilistic model (PLSI), which computes the textual similarities among the documents [30]. 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[2,20]. Textual similarities can also be used to retrieve traceability links between documentation and source code (e.g., [29,31–34]).

2.2. Identifying duplicate bug report with IR and topic modeling

Bug reports are submitted to issue tracking systems (e.g., Bugzilla[1]) by developers that use external libraries and APIs. For example, developers may report an issue in case of crashes or software malfunctioning. Since bug reports (or issues) are reported occasionally and on a voluntary basis, it is very common that the same issues (e.g., bug) are reported multiple times but by different users. Identifying duplicates in issue tracking systems is crucial to reduce the maintenance efforts from developers [2]. For example, newly reported issues might already be fixed in other versions of the software project. Hence, developers need to analyze both upcoming reports and old reports to avoid that the same issue is fixed/handled multiple times [2]. Besides, duplicate reports for issues that are still open (e.g., not fixed) provide complementary information that can be useful for developers during the bug fixing process [2,35].

Tables 1 and 2 reports two examples of duplicate bug reports for the `apache commons collections`. The two reports have been submitted independently by two users, for two different versions of the library and over a period of three years. Each reported issue contains multiple information, including the issue ID, a summary (short description of the issue), a complete description (which often includes code snippets), the version of the library affected by the issue, and the type of issue (e.g., bug). When a newly reported issue is submitted (issue #315 in our case), developers need to check past reported issues manually (e.g., issue #219), analyze the text written by the submitters, and decide whether to mark the issue as duplicate or proceed with debugging and fixing.

Table 1
Example of reported bug (#315) for `apache common collections` version 3.2.1.

ID	315
Open date	2009-02-03 19:30:04
Summary	`CollectionUtils.removeAll` calls the wrong `ListUtils` method
Description	Using version 3.2.1 as downloaded from maven's public repository. `CollectionUtils.removeAll` should call `ListUtils.removeAll` instead of `ListUtils.retainAll`. Currently {{ `public static Collection removeAll(Collection collection, Collection remove) { return ListUtils.retainAll(collection, remove); }` }} Suggested {{ `public static Collection removeAll(Collection collection, Collection remove) { return ListUtils.*removeAll*(collection, remove); }` }}
Version	3.2.1
Type	Bug

[1] https://www.bugzilla.org

Table 2
Example of reported bug (#219) for `apache common collections` version 3.2.2.

ID	219
Open date	2006-08-02 17:37:52
Summary	The `CollectionUtils.removeAll` method calls the `ListUtils.retainAll` method instead of the `ListUtils.removeAll` method
Description	The `CollectionUtils.removeAll` method calls the `ListUtils.retainAll` method instead of the `ListUtils.removeAll` method
Version	3.2.2
Type	Bug

While for small software projects manually classifying bug reports may not be unfeasible, larger projects can have hundreds of reported issues per day. For example, Anvik et al. [36] wrote that Eclipse received around 200 bug reports per day in 2005. In 2007, Mozilla received approximately 300 reports per day [37]. Therefore, semi-automated methods are needed to support developers in identifying duplicate bug reports.

Information retrieval and topic model techniques are powerful tools that can be applied to bug reports as well. This is because bug reports can be viewed as textual documents, and bug reports with large textual similarity have a higher probability of being duplicated. For example, Nguyen et al. [2] combined IR and topic models to detect duplicate reports semi-automatically. Hindle et al. [20] showed that continuously querying bug reports helps developers discover duplicates when submitting new bug reports. Sun et al. [38] applied discriminative models based on IR methods to separate duplicate bug reports and non-duplicate ones. Tian et al. [39] combined IR methods with machine learning (ML) techniques.

Textual similarities are computed using a distance function upon the computation of the term-by-document matrix (or its low-dimensional approximation produced by LDA). Each bug report is used as a query to retrieve the corresponding duplicated (past) reports. Therefore, the candidate list for each query is determined by sorting the past report in descending order of dissimilarity, which is computed using a *distance function*. Effective IR methods or topic models should assign better rankings to duplicate reports over non-duplicate ones. Many alternative distance functions can be used, such as the Euclidean distance, the cosine distance, the Hellinger distance, and so on. The distance to apply depends on the algebraic model used to represent the documents in the term-by-document matrix. For example, in the vector space model (VSM), documents correspond to vectors (the columns vector in the matrix) in a multi-dimensional space. In such a space, the document distances (or similarities) are computed by measuring the cosine of the angle formed by each pair of document vectors [34] (cosine distance). Instead, probabilistic models rely on distance functions based on probability theory (e.g., the Hellinger distance and the Jensen-Shannon Divergence [34]).

2.3. Latent dirichlet allocation

Latent Dirichlet Allocation (LDA) [7] is a generative probabilistic model for a collection of textual documents (corpus). It is a three-level hierarchical Bayesian model that associates documents with multiple topics [7]. In LDA, a topic is a cluster of relevant words in the corpus under analysis. 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 in Section 2.1. LDA generates two distributions of probabilities, one associated with the documents and the other one related to 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 technique if 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:

$$\underset{m\times n}{M} \approx \underset{m\times K}{\Phi} \times \underset{K\times n}{\Theta} \tag{2}$$

In Eqn 2 K is typically smaller than m. Using Θ, documents are clustered based on the topics they share and 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 corpus. `VEM` applies a deterministic variational EM method using expectation maximization [40]. The fast-collapsed `Gibbs sampling` generative model is an iterative process that applies a Markov Chain Monte Carlo algorithm [41]. In this paper, we focus on `Gibbs-sampling` as prior studies showed that it much faster [42], it can achieve more stable results [11], and better convergence towards the global optimum than VEM [8] for SE documents.

There are four hyperparameters to set when using the Gibbs sampling generative model for LDA [9,43]:

- The number of topics K to generate from the corpus;
- The hyperparameter α, which influences the distribution of the topics per document. Smaller α values lead to fewer topics per document.
- The hyperparameter β, which 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.

2.3.1. 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 corpus (*type-I instability*). Previous studies (e.g., [8,44]) suggested different strategies to increase LDA stability, including using random seeds, applying multiple Gibb restarts, or running LDA numerous times.

The Gibbs sampling generative method is a stochastic method that performs random samples of the corpus. 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 is using the same initial `seed` when running LDA with the same hyperparameters and for the same textual corpus.

Another well-known strategy to address *type-I instability* is restarting the Gibbs sampling to avoid converging toward local optima. For example, Hughes et al. [44] proposed a sparsity-promoting restart and observed dramatic gains due to the restarting. Binkley et al. [45] run the Gibbs sampler multiple times, suggesting that it reduces the probability of getting stuck in local optima. Recently, Mantyla et al. [46] performed multiple LDA runs and combined the results of different runs through clustering.

Agrawal et al. [8] observed that different documents orderings in the collections could also lead to slightly different topics (*type-II instability*). This type of instability can be addressed by running LDA multiple times, each time with various documents ordering. Then, LDA's stability with a given configuration can be measured by looking at the topics consistency across the independent LDA runs. In other words, the consistency of the topics is used as the surrogate metric to tune LDA with

meta-heuristics [8].

In this paper, we use both fixed `seeds` for the sampling and the restarting strategy. More details are provided in Section 4.5. Besides, we assess the surrogate metric proposed by Agrawal et al. [8] to address the *type-II instability*.

3. Automated tuning for LDA

When using LDA, a general problem is deciding the hyperparameters values to adopt when applying it to a specific corpus. Researchers from different communities agree that there is no universal setting that works well for any dataset (e.g., [9,44,45]). Researchers have proposed different heuristics to find (near) optimal hyperparameters for a given task [8,9,11–13]. Most of the early approaches focused on the number of topics K to set while using fixed values for α, β, and N [11–13]. Instead, later studies focus on tuning multiple LDA parameters [8,9,17]. The next subsections describe the heuristics (surrogate metrics) used to tune LDA via unsupervised optimization.

3.1. Topic coherence

In the context of generic textual corpora, the state-of-the-art intrinsic metric for LDA is the *topic coherence* proposed by Mimno et al. [15].

Definition 1. Let $df(t)$ be the *document frequency* of the word t. Let $D(t, t^*)$ be the co-document frequency of words t and t^*, i.e., the number of documents containing both words t and t^*. Let $T^{\tau} = t_1^{\tau}, \cdots, t_M^{\tau}$ be the list of M most probable words in the topic τ. The topic coherence is defined as [15]:

$$C(\tau, V^{\tau}) = \sum_{m=2}^{M} \sum_{l=1}^{m-1} \log \frac{D\left(t_m^{\tau}, t_l^{2}\right) + 1}{df(t_l^{\tau})} \tag{3}$$

In Eqn 3, $C(\tau, V^{\tau})$ measures the coherence of the topic τ as the sum of the pairwise distributional similarity for the most M probable words in the topic [47]. In other words, coherent topics should contain words that co-occur in the same documents within the collection. The overall coherence for an LDA model (with parameters $[K, N, \alpha, \beta]$) can then be computed as the arithmetic mean of the topic coherence scores of the K generated topics.

Mimno et al. [15] showed that LDA configurations that exhibit better topic coherence also generate topics that better matches the expert annotations. The study focused on a benchmark of 300,000 grant and journal paper abstracts from the National Institutes of Health and manually annotated by experts.

However, the following question still remains unanswered: *does topic coherence used for traditional corpora work for software documents as well*? This article investigates and compares the topic coherence with other metrics proposed in the software engineering literature

3.2. Silhouette coefficient

In the context of SE, Panichella et al. [9] used an internal metric for cluster quality analysis to estimate the fitness of LDA configurations. The intuition is that LDA can also be seen as a clustering algorithm. More specifically, they used the *silhouette coefficient* (fitness function) to guide genetic algorithms, which were used to find LDA hyperparameters that increased the coefficient values.

Definition 2. The *silhouette coefficient* for a cluster C is defined as [9]:

$$s(C) = \frac{1}{n} \sum_{i=1}^{n} s(d_i) \quad \text{with} \quad s(d_i) = \frac{b(d_i) - a(d_i)}{\max(a(d_i), b(d_i))} \tag{4}$$

In Eqn 4, $s(d_i)$ denotes the silhouette coefficient for the document d_i in the corpus; $a(d_i)$ 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 of the document d_i to another document in a different cluster (cluster separation); $s(C)$ measures the overall silhouette coefficient as the arithmetic mean of the coefficients $s(d_i)$ for all documents in the corpus. $s(C)$ 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. [9] showed that hyperparameters that increased the silhouette coefficient also lead to better external performances, such as precision and recall in traceability recovery. Besides, the LDA configurations found with GAs achieve an average precision that is pretty close to the global optimum. The silhouette coefficient and GA were also used in a later study [30] to configure the whole IR process (including the pre-processing) automatically.

3.3. Raw score for LDA stability

Recently, Agrawal et al. [8] further investigated the challenges of configuring LDA with search algorithms. They proposed an alternative surrogate metric, namely the *raw score*, to measure the quality of LDA settings. They used Differential Evolution (DE) rather than GAs. Their results showed that this evolutionary algorithm could generate optimal hyperparameter values that lead to more stable LDA models (topics and mixtures).

The raw score is computed by (1) shuffling the documents within the collection, (2) running LDA multiple times over different random shuffles of the collections; (3) computing the raw score $\mathbb{R}_M$ as the median number overlaps of size M words across the multiple runs [8]. To avoid any sampling bias, the score is evaluated on every $R = 10$ runs of LDA with 10 different documents shuffles. In this study, we consider the $\mathbb{R}_7$ score, which measures the median number overlaps of size $M = 7$ words. We selected $M = 7$ as it corresponds to the number of items (e.g., words) humans keep in the short memory on average [8,48].

An empirical comparison between GA and DE with the raw score showed that the latter needs fewer generations and produces more stable LDA models than the former. However, in [8], GA and LDA were configured with different termination criteria: a few dozens fitness evaluations for DE and thousands of fitness evaluations for GA. Besides, Agrawal et al. [8] did not use standard strategies (e.g., restarting strategies) to produce stable results for both GA and DE. Based on the results in [8], Mantyla et al. [46] used DE in combination with multiple LDA runs to achieve even more stable topics.

3.4. Motivations to the current study

Using LDA without parameter tuning can generate *low-quality* topics, i.e., topics that are not representative of the textual content of a given corpus. In the context of duplicate bug reports, LDA is used to rank past bug reports in issue tracking systems by their topic similarity to the new report used as the query. *Low-quality* topics mean that developers will need to analyze many past bug reports before identifying the duplicate ones. Therefore, generating high-quality topics is fundamental for a successful application of topic modeling.

When it comes to deciding which method to use for tuning LDA, practitioners (developers and researches) can choose different alternatives from the related literature. Prior studies advocated for different surrogate metrics to measure/predict the performance of LDA configurations in unsupervised tuning. However, different metrics have been assessed either on different corpora or evaluated separately. *Which alternative metric is the most suited for bug reports and SE artifacts in general?* Answering this question if fundamental, as SE documents differ from natural language corpora typically used in topic modeling and information retrieval [5].

Similarly, given a particular surrogate metric, which meta-heuristics or optimization algorithm to use is also *an open question. Does tuning LDA*

need intelligent search over random search? If yes, *shall we use local optimizers or global optimizers?* Besides, within the same category or optimizers, *do we need sophisticated variational operators (e.g., adaptive covariance matrix)?*

This paper sheds light on these open questions. More specifically, we compare the performance of seven different meta-heuristics (not only DE and GA) and surrogate metrics when configuring LDA for duplicate bug report identification. We consider two critical dimensions for practitioners: (1) the ability of LDA to identify duplicate bug reports with minimal human effort, and (2) the time needed to configure/tune LDA.

4. Empirical study

Our study aims to investigate whether some search-based approaches are superior to others for tuning LDA. The tuning can be performed using different combinations of meta-heuristics and surrogate metrics that guide the search in an unsupervised manner. The *perspective* is of researchers interested in evaluating the *effectiveness* of different tuning strategies for LDA and developers who want to identify duplicate bug reports with less effort. Therefore, 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 LDA produces better accuracy in identifying duplicate bug reports when tuned using specific meta-heuristics. For this RQ, we consider the different fitness functions (e.g., topic coherence) separately.
- **RQ2**: *What is the best surrogate metric for tuning LDA?*With this research question, we want to compare three state-of-the-art surrogate metrics defined for tuning LDA: (1) topic coherence, (2) silhouette coefficient, (3) raw score. In particular, we run the same meta-heuristics with each surrogate metric separately.
- **RQ3**: *Does the running time differ across the experimented meta-heuristics?*Related work [8] advocated using Differential Evolution (DE) over other meta-heuristics because it requires less running time. With our third research question, we aim to identify meta-heuristics that are more efficient than others in tuning LDA.
- **RQ4**: *What is the overhead of computing the different surrogate metrics?* Different surrogate metrics analyze different aspects of the topics that LDA generates. With this last research question, we aim to understand whether some surrogate metrics are less expensive to compute than others. Knowing which metrics are more expensive will help us to give recommendations to developers for practical usage.

4.1. Benchmark

The benchmark of our study consists of ten projects from the `Bench4BL` corpus [6] and publicly available in GitHub[2]. Lee et al. [6] used the benchmark to perform a comprehensive reproduction study of state-of-the-art IR-based bug localization techniques. For our study, we selected ten Java projects from `Bench4BL`: five projects from the *apache commons library*[3], four projects from Spring[4], and one project from JBoss[5]. The characteristics of the selected projects are reported in Table 3. We chose these ten projects because they have been widely used in the SBSE literature (e.g., [49] for the apache libraries) and are well-maintained together with issue tracking systems.

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

Table 3
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	1617	245	8 (3%)
Apache hive	4651	1241	270 (22%)
Spring Datacmns	604	158	15 (9%)
Spring Roo	1109	714	72 (10%)
Spring Sec	1618	541	68 (13%)
Spring Spr	6512	130	73 (56%)
JBoss WFly	8990	984	27 (3%)

For each project, the `Bench4BL` contains (i) issues (from their issue tracking systems) that are explicitly labeled as `bugs` by the original developers, and (ii) the corresponding patches/fixes [6]. 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 duplicates (i.e., the oracle) have been identified and marked by the original developers in the issue tracking systems. The percentage of duplicated bug reports ranges between 3% for *apache commons math* and 56% for *Spring SPR*.

4.2. Meta-heuristic selection

Meta-heuristics are applied to search for LDA configurations that optimize the surrogate metrics described in Section 3. A candidate solution (i.e., LDA configuration) is a numerical vector of size four, whose entries correspond to the parameters described in Table 4. The table also describes the upper and lower bounds for each LDA parameter.

In this paper, we selected seven state-of-the-art meta-heuristics, namely *Genetic Algorithms* (GAs), *Differential Evolution* (DE), *Particle Swarm Optimization* (PSO), *Simulated Annealing* (SA), *Random Search* (Ran), *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES), and *Stochastic Hill Climbing* (SHC). The meta-heuristics are representative of different categories of meta-heuristics, such as local (SA, SHC), global (GA, PSO), population-based (PSO, DE, GA), adaption-based (CMA-ES), and non-evolutionary (Ran) optimization methods. The next subsections describe the different meta-heuristics and their main characteristics.

4.2.1. Random search

Random 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 [50], and it is often used as a baseline in SBSE.

4.2.2. Stochastic hill climbing

SHC is a local search algorithm that iteratively evolves one single solution. It starts with a random solution and explores the neighborhood in an attempt to find nearby better solutions. The traditional Hill Climbing explores all possible neighbors to decide which path the follow in the search space. However, exploring all potential neighbors can be very expensive (e.g., for high-dimensional problems) or infeasible (e.g., for continuous numerical problems). In this paper, we opted for SHC, which randomly generates only one neighbor decides (based on the

Table 4
List of LDA parameters and their ranges.

Parameter	Range	Type	Description
K	[10, #documents]	Integer	Number of topics to extract
N	[200, 300]	Integer	Number of Gibbs iterations
α	[0, 1]	Double	Distribution of topics over documents
β	[0, 1]	Double	Distribution of topics over words

magnitude of the improvement) whether to update the current solution or not in each iteration. For the neighborhood exploration, we use the *polynomial mutation* proposed by Deb and Agrawal for numerical problems.

SHC is faster than population-based algorithms to converge toward an optimal solution for continuous problems [51,52]. Since SHC only accepts better neighbors, it might converge toward a local optimum rather than the global one(s) [52]. A simple strategy to avoid premature convergence is *restarting* the search multiple times. However, this strategy is not applicable to problems with expensive fitness evaluations where only a few evaluations can be performed (e.g., when tuning LDA).

4.2.3. Simulated annealing

SA is a meta-heuristic that also evolves only one solution at a time [53]. SA improves upon HC by accepting worse neighbors with a certain probability. This downhill move reduces the chances of getting stuck in local optima [52]. SA works as follows: 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 worse 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.

4.2.4. Genetic algorithms

GAs have been used in a prior study to configure LDA [9] and the whole IR process [30]. GA is a population-based meta-heuristic that evolves a pool of randomly-generated solutions (LDA configurations) through subsequent generations. In each generation, solutions are selected based on their fitness values (e.g., silhouette coefficient) using the *binary tournament selection*. Fittest solutions (parents) are combined using *binary-simulated crossover* and *polynomial mutation* [54] to form new solutions (offspring). Then, the population for the new generation is formed by selecting the best solutions among parents and offspring (*elitism*). GA is very effective in global optimization problems or problems with a complex fitness landscape. The population samples multiple regions of the search space; this reduces the probability of getting stuck in local optima compared to local solvers (e.g., HC). GAs have been widely used in the literature to solve multiple types of problems, such as continuous, discrete, combinatorial, and many-objective problems. However, they can fail to find the global optimum in problems with deceptive, ragged, and noisy landscapes [55].

4.2.5. Differential evolution

DE is an evolutionary algorithm used by Agrawal et al. [8] for tuning LDA. 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 hyperparameters). The differential operator is applied with a probability $p_c \in [0; 1]$ (crossover probability).

The existing literature showed that DE is very competitive compared to other evolutionary algorithms. The strength comes from the fact that new solutions are created using the "difference" between the genes in the solutions/chromosomes. While DE is often reported to be very effective and efficient, it also comes with some limitations [56]. First, DE can only be applied to continuous problems (e.g., the genes with real numbers) and is not robust in problems with strong interdependency of the genes [56].

4.2.6. Particle swarm optimization

PSO was proposed by Eberhart and Kennedy [57]. Similar 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 by updating their *positions* and *velocity*. The position and velocity of each particle are guided by its own best-known position in the search-space as well as the best-known position for the entire pool (swarm). More specifically, the velocity is updated in each iteration using the formula:

$$v = w \times v + c_1 r_1 + (p_{best} - x) + c2 r_2 (g_{best} - x) \quad (5)$$

In Eqn 5, c_1 and c_2 are the search weights; r_1 and r_2 are randomly generated number in [0; 1]; p_{best} is the best know position for the given particle, while g_{best} denotes the best know position for the entire warm.

PSO has been widely used in many problems, including constrained, discrete, and continuous problems [58]. It is also reported to be faster than GAs for specific problems [58]. However, PSO has limitations in dynamic environments and can be easily trapped in local optima [59].

4.2.7. Covariance matrix adaptation evolution strategy

CMA-ES is an evolutionary strategy proposed by Hansen and Ostermeier [60] for continuous problems. CMA-ES starts with a random population of solutions, which are evaluated according to the fitness function. CMA-ES selects the best solutions within the population and estimates the local covariance matrix of the objective function. Such a matrix is used to sample new solutions in the next iterations such that the evolution continues toward the successful search paths discovered in early iterations. CMA-ES is known to be a very efficient and fast algorithm for numerical problems as it approximates the gradient of the objective function without using derivatives. Furthermore, CMA-ES is very robust to linear transformations, such as rotations. On the other hand, CMA-ES requires many data points (i.e., samples) to estimate the covariance matrix properly.

4.3. Parameter settings

For the search, we opt for the standard parameter setting and search operators suggested in the literature [8,9,61–63]. In particular, we used the following parameter values:

- *Genetic Algorithms*: population size of 10 LDA configurations [8]; *binary simulated crossover* with probability p_c=0.9 [64]; *polynomial mutation* with mutation probability p_m=0.25 (i.e., $1/n$, where n is the number of hyperparameters for LDA) [9], and *mutation index* η_m=20 [61].
- *Differential Evolution*: population size μ=10 [8] (the same as GAs); *differential weight factor* $f = 0.8$; crossover probability p_c=0.9. Note that the last two parameter values are recommended in the literature [62].
- *Simulated Annealing*: neighbors are generated using the *polynomial mutation* with a probability p_m=0.25; the *number of steps* per temperature *ns*=10; the *number of temperatures nt*=5. Note that we use the same mutation operator used in GA with the same setting. For the number of steps and temperature, we had to adjust it to the low number of fitness evaluations that can be done for expensive functions, such as the surrogate metrics for tuning LDA [8].
- *Particle Swarm Optimization*: population size μ=10; Zheng et al. [63] recommended to use a value $w_i \in [0.4; 0.9]$ for the *inertia weight*; thus, we choose a value of w_i=0 as in our trials provided the best results. We also use the *search weights* c_1=c_2=1, which are the default values in the library we used for the implementation (see Section 4.4).

- *Stochastic Hill Climbing*: the only parameter in SHC is the function used to explore the neighborhood of the initial solution. We opted for the same mutation operator used for the other meta-heuristics, i.e., the *polynomial mutation* with a mutation probability p_m=0.25.
- *CMA-ES*: we set the population size of μ=10 LDA configurations. Other parameters are updated in an adaptive manner by the algorithm itself.
- *Random*: 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. [9] used GA with 100 individuals and 100 generations, corresponding to 10K FEs; Agrawal et al. [8] used DE with ten individuals and three generations, corresponding to 30 FEs. Agrawal et al. [8] 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 is very expensive for large corpora. Based on the motivation by Agrawal et al. [8]., 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 [8] used fewer FEs only for DE.

4.4. Implementation

For LDA, we use its implementation available in the package `topicmodels` in `R` [65]. We chose this implementation over other implementations (e.g., `Mallet`[6] in Java) because it provides an interface to the original LDA implementation in `C` code by Blei et al. [7]. Furthermore, Binkley et al. [45] 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 [9]. It also supports random restarts to achieve stable LDA models. We used the implementations of the meta-heuristics available in `R`:

- Real-coded genetic algorithms from the package `GA` [66].
- Differential evolution from the package `DEoptim` [67].
- Random search from the package `randomsearch` [68].
- Simulated-Annealing, Particle Swarm Optimization, and Stochastic Hill Climbing from the package `NMOF` [69].
- CMA-ES from the package `parma` [70]

The `R` scripts and datasets used in our experiment are publicly available at the following link: http://doi.org/10.5281/zenodo.4016590.

4.5. Experimental methodology

For each project, and for each fitness function, we run each meta-heuristic 100 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 (detailed below). At the end of each run, we use the LDA configuration produced by the meta-heuristic and fitness function under analysis, and we generated the corresponding LDA model, and the *topic-by-document matrix* (Θ) in particular.

To answer RQ1 and RQ2, we use the TOP_k metric, which measures the performance of a retrieval 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]:

$$TOP_k(Q) = \frac{1}{|Q|}\sum_{q \in Q} in_k(i) \quad (6)$$

In Eqn 6, $in_k(i)$ 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 value, the better the performance of LDA with a given configuration.

Previous work [20,38,71] on duplicate bug report identification used the cut-offs k=5 and k=10. However, a more recent study [72] recommended deeper cut-offs as they "*offer better robustness to sparsity and popularity biases than shallower cut-offs.*" In this paper, we consider four values of k, i.e., TOP_5, TOP_{10}, TOP_{15}, and TOP_{20}. The first two cut-offs (k=5, 10) are those used in the literature for bug reports, while the last two values (k=15, 20) are chosen based on the recommendation of using deeper and more robust cut-offs.

To answer RQ3 and RQ4, we compare the running time required by the different meta-heuristics, and each fitness function to terminate the search process in each independent run. For our analysis, we compare the median running time across the 100 independent runs and the interquartile range. The interquartile range (IQR) is a measure of statistical dispersion, and it corresponds to the difference between the third (75th percentiles) and the first (25th percentile) quartiles of a given distribution, e.g., running time over 100 runs in our case. To graphically compare the different distributions, we also use box-plots. A box-plot graphically depicts a distribution through its quartiles. The box represents the middle 50% of the distribution, i.e., first, second, and third quartiles. Whiskers depict data points that are outside the first and third quartiles.

To assess the statistical significance, we followed the literature guidelines [21] for comparing meta-heuristics over a set of benchmark test problems (projects in our case). The Friedman test is the non-parametric equivalent to ANOVA with unreplicated block design. Differently from ANOVA, it does not require that the data follow a normal distribution. The Friedman test is used to assess the significance of the difference in treatments (meta-heuristics and fitness functions in our case) across multiple test attempts (ten projects in our case).

In our study, each combination of meta-heuristic and fitness function (F, H) produced 4 (TOP_k metrics) $\times$ 10 (projects) $\times$ 100 (runs) = 4000 data points. For statistical analysis, we consider the median TOP_k scores across the 100 runs, resulting in 28 average scores per combination of meta-heuristic and fitness function (F, H). Then, we compare the obtained distributions using the Friedman test with the significance level $\alpha = 0.05$ [21].

While the Friedman test reveals whether some treatments significantly differ from the others, it does not tell for which combination of treatments (F, H) such significance holds. To further understand which combination (F, H) performs better than others, we use the Nemenyi test. The Nemenyi test measures the difference across treatments by computing the average rank of each treatment across multiple projects [22]. The tuning strategy with the lower (best) average rank is preferable over the ones with larger (worse) average ranks. Two tuning strategies A_1 and A_2 are significantly different if their corresponding average ranks differ by at least the given *critical distance* (*CD*) [22].

In total, we performed 10 (projects) $\times$ 7 (meta-heuristics) $\times$ 3 (fitness functions) $\times$ 100 (independent runs) = 21,000 LDA tunings. The experiment took six months of running time on a dedicated server with the following specifications: Ubuntu 18.04.4 LTS with Linux kernel 4.4.0, on a 32-core Intel(R) Xeon(R) Gold 5217 CPU at 3.00GHz, and with 128GB of RAM.

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

4.6. Addressing the instability of LDA

In this paper, we address the stability problem using two standard strategies: seeding and random restart (*type-I stability*). When evaluating each LDA configuration (individual), we store both the fitness function value 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 (fitness function, topics, and mixtures) even when LDA is re-run multiple times with the same hyperparameters. 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 $n = 3$ times (independent runs), and the generated topics and mixtures are obtained by averaging the results achieved across the independent results. This restarting procedure is implemented in the `topicmodels` package available in `R`.

With regards to *type-II instability*, we used the raw score as one of the alternative fitness function in our study. As explained in Section 3.3, the raw score avoids *type-II instability* by running LDA multiple times, each time with a different document ordering. When using the raw score as the fitness function, we disable the restarting strategy in `topicmodels` to avoid double resampling. Indeed, Gibbs is already restarted more times to compute the words overlap during the computation of the raw score.

5. Empirical results

This section discusses the empirical results, addressing, one by one, our research questions.

5.1. Comparing the meta-heuristics

Results with the silhouette coefficient. Table 5 shows the median and the interquartile range (IQR) of the performance scores (TOP_5, TOP_{10}, TOP_{15}, and TOP_{20}) achieved by the meta-heuristics when using the *silhouette coefficient* by Panichella et al. [9] as the fitness function. The results are collected over 100 independent runs. First, we can notice that there is no "master" (dominant) meta-heuristic that outperforms the others for all software projects when using the silhouette coefficient (SC). SHC, GA, and SA produce the best (largest) TOP_k scores for different projects and different k values. GA achieves the highest TOP_k in 21 out of 40 median values in Table 5: in two projects for TOP_5, six projects for TOP_{10}, six projects for TOP_{15}, and seven projects for TOP_{20}. For the projects `hive`, and `sec`, GA outperforms all other meta-heuristics according to TOP_{10}-TOP_{20} and TOP_{15}-TOP_{20} score, respectively. The differences with the second-highest scores range between 1% (`hive` with TOP_5) and 4% (`hive` with TOP_{10}).

SHC achieves the highest TOP_k scores in 21 out of 40 median values in Table 5. The difference with the second-highest scores is 4% (`io` with TOP_5). For other projects, SHC either ties with the best performing meta-heuristics (20/40 cases) or reaches lower TOP_k scores than other meta-

Table 5
Median and IQR of the scores achieved by the evaluated meta-heuristics when using the Silhouette Coefficient (SC) as fitness function (FF).

System	Metric	CMA-ES		DE		GA		SHC		PSO		Ran		SA	
		Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR
Collections	TOP.5	0.86	0.19	0.90	0.10	0.95	0.14	0.95	0.10	0.90	0.14	0.90	0.10	1.00	0.10
	TOP.10	0.90	0.15	0.95	0.10	1.00	0.10	1.00	0.10	0.95	0.10	0.95	0.10	1.00	0.10
	TOP.15	0.90	0.19	0.95	0.10	1.00	0.10	1.00	0.10	0.98	0.10	0.95	0.10	1.00	0.05
	TOP.20	0.90	0.14	1.00	0.10	1.00	0.05	1.00	0.10	1.00	0.10	1.00	0.10	1.00	0.05
Datacmns	TOP.5	0.34	0.24	0.38	0.17	0.38	0.14	0.45	0.17	0.41	0.15	0.34	0.17	0.45	0.14
	TOP.10	0.48	0.25	0.62	0.14	0.62	0.14	0.62	0.10	0.62	0.17	0.59	0.14	0.62	0.14
	TOP.15	0.55	0.28	0.72	0.14	0.72	0.14	0.69	0.11	0.69	0.14	0.69	0.14	0.69	0.14
	TOP.20	0.57	0.31	0.76	0.10	0.76	0.14	0.72	0.14	0.72	0.14	0.72	0.14	0.72	0.14
Hive	TOP.5	0.15	0.15	0.32	0.15	0.33	0.05	0.32	0.05	0.31	0.08	0.29	0.11	0.31	0.03
	TOP.10	0.18	0.18	0.39	0.12	0.41	0.05	0.38	0.05	0.40	0.06	0.38	0.14	0.38	0.06
	TOP.15	0.20	0.20	0.40	0.10	0.45	0.05	0.41	0.04	0.45	0.07	0.41	0.11	0.40	0.06
	TOP.20	0.22	0.21	0.40	0.12	0.45	0.05	0.43	0.05	0.47	0.07	0.44	0.10	0.42	0.04
Io	TOP.5	0.54	0.23	0.54	0.08	0.54	0.10	0.58	0.08	0.54	0.15	0.54	0.15	0.54	0.08
	TOP.10	0.62	0.23	0.69	0.08	0.69	0.15	0.69	0.08	0.69	0.15	0.69	0.23	0.69	0.08
	TOP.15	0.62	0.15	0.69	0.08	0.77	0.08	0.69	0.08	0.73	0.10	0.77	0.15	0.69	0.15
	TOP.20	0.69	0.12	0.77	0.15	0.77	0.15	0.73	0.08	0.77	0.15	0.77	0.15	0.77	0.08
Lang	TOP.5	0.40	0.22	0.50	0.24	0.48	0.26	0.55	0.15	0.48	0.19	0.49	0.24	0.57	0.15
	TOP.10	0.55	0.27	0.67	0.17	0.64	0.15	0.67	0.08	0.67	0.12	0.64	0.12	0.67	0.10
	TOP.15	0.62	0.29	0.71	0.12	0.71	0.12	0.71	0.07	0.71	0.10	0.69	0.12	0.71	0.10
	TOP.20	0.67	0.29	0.76	0.10	0.76	0.12	0.74	0.07	0.74	0.07	0.74	0.10	0.74	0.10
Math	TOP.5	0.44	0.27	0.44	0.25	0.44	0.25	0.50	0.19	0.50	0.27	0.44	0.19	0.50	0.19
	TOP.10	0.56	0.25	0.62	0.27	0.62	0.20	0.69	0.19	0.69	0.19	0.62	0.25	0.69	0.19
	TOP.15	0.62	0.25	0.75	0.25	0.75	0.19	0.75	0.19	0.75	0.25	0.75	0.25	0.69	0.19
	TOP.20	0.62	0.25	0.75	0.19	0.75	0.25	0.75	0.12	0.75	0.19	0.81	0.19	0.75	0.19
Roo	TOP.5	0.18	0.11	0.25	0.13	0.24	0.16	0.29	0.09	0.27	0.09	0.27	0.07	0.29	0.08
	TOP.10	0.22	0.16	0.34	0.11	0.34	0.14	0.37	0.08	0.35	0.08	0.34	0.05	0.37	0.06
	TOP.15	0.24	0.18	0.37	0.11	0.38	0.10	0.40	0.07	0.38	0.09	0.37	0.08	0.40	0.05
	TOP.20	0.26	0.20	0.39	0.11	0.40	0.07	0.42	0.05	0.40	0.08	0.39	0.10	0.42	0.05
Sec	TOP.5	0.33	0.18	0.44	0.14	0.48	0.12	0.46	0.05	0.46	0.09	0.44	0.12	0.48	0.07
	TOP.10	0.43	0.21	0.54	0.15	0.58	0.08	0.58	0.07	0.56	0.09	0.56	0.12	0.57	0.06
	TOP.15	0.48	0.22	0.59	0.15	0.63	0.07	0.60	0.07	0.59	0.10	0.59	0.11	0.60	0.07
	TOP.20	0.50	0.23	0.62	0.15	0.65	0.08	0.62	0.08	0.61	0.12	0.63	0.12	0.61	0.08
Spr	TOP.5	0.56	0.12	0.59	0.11	0.61	0.14	0.67	0.05	0.61	0.07	0.54	0.13	0.69	0.08
	TOP.10	0.69	0.12	0.75	0.06	0.76	0.04	0.74	0.07	0.75	0.05	0.72	0.05	0.76	0.06
	TOP.15	0.73	0.13	0.80	0.05	0.80	0.04	0.78	0.09	0.81	0.03	0.77	0.05	0.80	0.05
	TOP.20	0.76	0.12	0.84	0.06	0.83	0.05	0.82	0.07	0.84	0.02	0.83	0.05	0.82	0.06
WFly	TOP.5	0.36	0.30	0.58	0.15	0.53	0.15	0.58	0.09	0.53	0.17	0.57	0.13	0.58	0.11
	TOP.10	0.42	0.30	0.64	0.15	0.59	0.12	0.66	0.06	0.60	0.17	0.60	0.11	0.66	0.06
	TOP.15	0.45	0.30	0.65	0.13	0.64	0.12	0.68	0.06	0.64	0.11	0.66	0.08	0.70	0.08
	TOP.20	0.47	0.32	0.65	0.13	0.65	0.13	0.70	0.04	0.68	0.11	0.66	0.11	0.70	0.06

heuristics. SHC achieves the highest TOP_k scores in 24 out of 40 median values in Table 5. In 22 of such cases, it achieves the same (best) TOP_k scores of the best performing meta-heuristics.

PSO and DE achieve the highest TOP_k scores in 11 and 13 cases, respectively. Therefore, *our results indicate that DE is not superior to other meta-heuristics*, as argued in prior studies, *at least when using the silhouette coefficient as the fitness function*. Random search yields the highest TOP_k in 5 out of 40 median values. Instead, CMA-ES never produces TOP_k scores that are comparable to the best performing meta-heuristic in the comparison. Therefore, it is not a suitable meta-heuristic for topic models, at least when using the *silhouette coefficient* and in the context of identifying duplicate bug reports.

Results with topic coherence. Table 6 reports the median and the interquartile range (IQR) of the performance scores (TOP_5, TOP_{10}, TOP_{15}, and TOP_{20}) when using the *topic coherence* proposed by Mimno et al. [15] as the fitness function. The results are collected over 100 independent runs.

As we can observe, even when using *topic coherence*, there is no "master" (dominant) meta-heuristic that outperforms the others for all software projects. However, we can observe that local solvers (i.e., SA and SHC) perform better than other meta-heuristics. Indeed, SA and SHC provide the best TOP_k scores in 20 and 17 cases out of 40 median values (see Table 6). The two meta-heuristics are equivalent (i.e., they produce exactly the same median TOP_k scores) in nine cases ($11/40 \approx 28\%$), while they differ in all other cases.

The results of the two local solvers (i.e., SA and SHC) for the project `WFly` are very notable: they both reach between +11% and +24% on TOP_5 compared to the alternatives. We also notice that SHC achieves the best TOP_k scores for the second-largest project, i.e., `hive`. Instead, SA produced the best TOP_k scores for the largest project, i.e., `WFly`. This indicated that local solvers might perform better than other meta-heuristics for large corpora when using the topic coherence.

The other meta-heuristics achieve the best TOP_k scores in a lower number of cases compared to the two local solvers. GA and DE yield the highest TOP_k scores in 15 out of 40 median values in Table 6. Random search and PSO do so in seven and 13 cases, respectively. Also, for *topic coherence*, CMA-ES produces the highest TOP_k score in only one case.

Results with the raw score. The results achieved by the meta-heuristics when using the *raw score* proposed by Agrawal et al. [8] are reported in Table 7. The table reports the median and the IQR of the performance scores (i.e., TOP_5, TOP_{10}, TOP_{15}, and TOP_{20}) collected over 100 independent runs.

Like the results achieved for the other two fitness functions, *we do not observe one single meta-heuristic dominating all the others for all projects in our benchmark*. More specifically, all meta-heuristics produce at least the highest TOP_k for at least one project (although with different k values).

To quantify, SHC achieves the best TOP_k scores in 26 cases out of 40, followed by SA with 24 cases, and GA with 19 cases. Once again, SHC and SA turn out to be the most performing meta-heuristics for the two largest projects in our benchmark, i.e., `hive`, and `wfly`. Random search and DE both produce the highest TOP_k scores in 13 cases (different projects with different k values for the TOP_k metric). CMA-ES yields

Table 6
Median and IQR of the scores achieved by the evaluated meta-heuristics when using the Coherence (Coh.) as fitness function (FF).

System	Metric	CMA-ES		DE		GA		SHC		PSO		Ran		SA	
		Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR
Collections	TOP.5	0.90	0.19	0.98	0.10	1.00	0.10	1.00	0.10	0.95	0.10	0.95	0.10	0.95	0.10
	TOP.10	0.95	0.14	1.00	0.10	1.00	0.10	1.00	0.10	0.95	0.10	0.95	0.10	1.00	0.10
	TOP.15	0.95	0.14	1.00	0.10	1.00	0.10	1.00	0.10	1.00	0.10	0.95	0.10	1.00	0.10
	TOP.20	0.95	0.10	1.00	0.10	1.00	0.10	1.00	0.10	1.00	0.10	0.95	0.10	1.00	0.10
Datacmns	TOP.5	0.38	0.17	0.34	0.14	0.38	0.21	0.45	0.14	0.38	0.18	0.31	0.16	0.48	0.17
	TOP.10	0.52	0.17	0.59	0.17	0.62	0.17	0.62	0.14	0.62	0.14	0.55	0.17	0.62	0.10
	TOP.15	0.55	0.25	0.72	0.14	0.72	0.14	0.67	0.14	0.72	0.11	0.69	0.14	0.69	0.14
	TOP.20	0.59	0.24	0.79	0.10	0.78	0.10	0.72	0.15	0.79	0.10	0.76	0.10	0.72	0.14
Hive	TOP.5	0.17	0.13	0.15	0.05	0.19	0.09	0.32	0.04	0.22	0.09	0.18	0.05	0.32	0.01
	TOP.10	0.22	0.16	0.21	0.08	0.27	0.12	0.39	0.05	0.31	0.12	0.24	0.06	0.39	0.02
	TOP.15	0.25	0.18	0.27	0.08	0.32	0.11	0.42	0.02	0.36	0.13	0.29	0.07	0.40	0.03
	TOP.20	0.26	0.20	0.32	0.09	0.38	0.11	0.44	0.03	0.41	0.12	0.33	0.08	0.43	0.02
IO	TOP.5	0.54	0.08	0.54	0.08	0.58	0.08	0.54	0.08	0.54	0.08	0.54	0.08	0.54	0.15
	TOP.10	0.62	0.15	0.65	0.08	0.69	0.08	0.69	0.08	0.62	0.08	0.69	0.08	0.69	0.08
	TOP.15	0.65	0.08	0.69	0.15	0.69	0.08	0.69	0.08	0.69	0.15	0.69	0.15	0.69	0.15
	TOP.20	0.69	0.15	0.69	0.08	0.69	0.08	0.77	0.08	0.69	0.08	0.69	0.08	0.69	0.08
Lang	TOP.5	0.52	0.19	0.62	0.12	0.60	0.10	0.60	0.14	0.60	0.10	0.62	0.10	0.57	0.14
	TOP.10	0.64	0.17	0.71	0.10	0.71	0.10	0.69	0.10	0.69	0.07	0.69	0.07	0.67	0.10
	TOP.15	0.69	0.17	0.74	0.10	0.74	0.10	0.71	0.10	0.71	0.10	0.71	0.07	0.71	0.10
	TOP.20	0.71	0.17	0.76	0.07	0.75	0.07	0.74	0.10	0.74	0.10	0.74	0.07	0.74	0.10
Math	TOP.5	0.44	0.19	0.56	0.12	0.59	0.12	0.56	0.14	0.56	0.12	0.56	0.12	0.50	0.19
	TOP.10	0.62	0.27	0.69	0.19	0.69	0.16	0.69	0.12	0.69	0.19	0.69	0.19	0.69	0.19
	TOP.15	0.69	0.25	0.69	0.19	0.69	0.16	0.75	0.19	0.69	0.19	0.69	0.12	0.69	0.19
	TOP.20	0.69	0.20	0.69	0.19	0.69	0.19	0.75	0.12	0.69	0.19	0.69	0.14	0.69	0.19
Roo	TOP.5	0.20	0.12	0.21	0.12	0.26	0.14	0.28	0.08	0.25	0.08	0.18	0.12	0.30	0.07
	TOP.10	0.27	0.17	0.35	0.15	0.38	0.16	0.37	0.05	0.36	0.07	0.28	0.15	0.38	0.06
	TOP.15	0.31	0.18	0.40	0.13	0.40	0.12	0.40	0.05	0.42	0.05	0.35	0.17	0.40	0.05
	TOP.20	0.34	0.19	0.44	0.10	0.43	0.06	0.43	0.05	0.44	0.06	0.40	0.17	0.42	0.05
Sec	TOP.5	0.41	0.20	0.47	0.08	0.47	0.08	0.47	0.07	0.47	0.08	0.47	0.08	0.48	0.07
	TOP.10	0.51	0.21	0.61	0.07	0.59	0.07	0.56	0.09	0.61	0.07	0.61	0.06	0.57	0.07
	TOP.15	0.55	0.22	0.66	0.05	0.65	0.06	0.60	0.07	0.66	0.05	0.66	0.04	0.60	0.07
	TOP.20	0.57	0.22	0.68	0.05	0.69	0.05	0.64	0.09	0.69	0.05	0.69	0.04	0.62	0.09
Spr	TOP.5	0.66	0.15	0.70	0.05	0.68	0.05	0.69	0.06	0.70	0.05	0.68	0.05	0.69	0.07
	TOP.10	0.75	0.16	0.74	0.06	0.74	0.06	0.77	0.06	0.74	0.05	0.73	0.05	0.76	0.06
	TOP.15	0.79	0.13	0.77	0.06	0.76	0.05	0.79	0.06	0.77	0.05	0.77	0.05	0.79	0.06
	TOP.20	0.80	0.14	0.78	0.06	0.77	0.05	0.81	0.05	0.78	0.05	0.78	0.05	0.81	0.06
WFly	TOP.5	0.42	0.25	0.38	0.19	0.44	0.19	0.58	0.08	0.47	0.23	0.34	0.23	0.58	0.10
	TOP.10	0.49	0.27	0.47	0.15	0.55	0.21	0.64	0.09	0.55	0.23	0.43	0.25	0.66	0.08
	TOP.15	0.57	0.26	0.53	0.16	0.61	0.18	0.66	0.09	0.60	0.21	0.47	0.21	0.68	0.06
	TOP.20	0.58	0.28	0.58	0.14	0.65	0.17	0.68	0.09	0.64	0.19	0.57	0.17	0.68	0.08

Table 7
Median and IQR of the scores achieved by the evaluated meta-heuristics when using the Raw Score ($\mathbb{R}_7$) as fitness function (FF).

System	Metric	CMA-ES		DE		GA		SHC		PSO		Ran		SA	
		Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR
Collections	TOP.5	0.90	0.24	0.95	0.10	1.00	0.10	1.00	0.10	0.95	0.10	0.95	0.10	0.95	0.10
	TOP.10	0.95	0.19	1.00	0.10	1.00	0.10	1.00	0.10	1.00	0.10	1.00	0.10	1.00	0.10
	TOP.15	0.95	0.14	1.00	0.10	1.00	0.10	1.00	0.10	1.00	0.10	1.00	0.10	1.00	0.10
	TOP.20	0.95	0.14	1.00	0.10	1.00	0.10	1.00	0.10	1.00	0.10	1.00	0.10	1.00	0.10
Datacmns	TOP.5	0.38	0.17	0.45	0.14	0.45	0.21	0.45	0.14	0.45	0.17	0.45	0.14	0.45	0.15
	TOP.10	0.55	0.18	0.62	0.10	0.60	0.15	0.60	0.14	0.62	0.14	0.62	0.14	0.62	0.14
	TOP.15	0.62	0.24	0.66	0.10	0.69	0.10	0.66	0.10	0.69	0.14	0.69	0.10	0.69	0.14
	TOP.20	0.67	0.21	0.69	0.17	0.72	0.11	0.72	0.14	0.72	0.11	0.72	0.10	0.72	0.10
Hive	TOP.5	0.26	0.13	0.19	0.16	0.25	0.14	0.32	0.03	0.19	0.20	0.16	0.10	0.31	0.03
	TOP.10	0.35	0.19	0.28	0.17	0.31	0.10	0.37	0.01	0.26	0.20	0.25	0.09	0.36	0.01
	TOP.15	0.38	0.22	0.35	0.14	0.34	0.05	0.40	0.01	0.30	0.17	0.32	0.08	0.39	0.01
	TOP.20	0.40	0.23	0.37	0.11	0.37	0.04	0.42	0.01	0.35	0.13	0.35	0.06	0.41	0.01
Io	TOP.5	0.54	0.15	0.54	0.08	0.54	0.15	0.54	0.08	0.54	0.15	0.54	0.08	0.54	0.08
	TOP.10	0.62	0.15	0.69	0.08	0.69	0.08	0.69	0.10	0.69	0.08	0.65	0.08	0.69	0.08
	TOP.15	0.69	0.15	0.69	0.15	0.69	0.15	0.69	0.15	0.69	0.15	0.69	0.15	0.69	0.15
	TOP.20	0.69	0.15	0.73	0.08	0.69	0.08	0.69	0.08	0.69	0.08	0.69	0.08	0.77	0.08
LANG	TOP.5	0.52	0.17	0.62	0.07	0.61	0.08	0.60	0.12	0.62	0.07	0.62	0.10	0.57	0.12
	TOP.10	0.64	0.14	0.67	0.05	0.67	0.07	0.69	0.07	0.68	0.10	0.67	0.10	0.69	0.12
	TOP.15	0.69	0.17	0.69	0.05	0.69	0.10	0.71	0.07	0.69	0.10	0.69	0.10	0.71	0.10
	TOP.20	0.71	0.19	0.69	0.10	0.69	0.10	0.74	0.07	0.71	0.07	0.71	0.07	0.74	0.07
MATH	TOP.5	0.44	0.19	0.56	0.19	0.56	0.12	0.53	0.19	0.50	0.19	0.56	0.19	0.50	0.19
	TOP.10	0.62	0.19	0.62	0.12	0.62	0.12	0.62	0.12	0.62	0.19	0.62	0.14	0.69	0.19
	TOP.15	0.62	0.19	0.62	0.19	0.62	0.12	0.69	0.12	0.62	0.19	0.62	0.19	0.69	0.19
	TOP.20	0.69	0.12	0.66	0.19	0.62	0.06	0.69	0.12	0.69	0.19	0.69	0.19	0.69	0.19
ROO	TOP.5	0.21	0.12	0.16	0.13	0.26	0.18	0.29	0.09	0.23	0.11	0.21	0.10	0.28	0.07
	TOP.10	0.31	0.18	0.27	0.10	0.34	0.15	0.36	0.07	0.31	0.10	0.31	0.12	0.36	0.06
	TOP.15	0.34	0.23	0.34	0.13	0.41	0.11	0.39	0.06	0.36	0.08	0.36	0.12	0.40	0.05
	TOP.20	0.38	0.25	0.38	0.11	0.42	0.09	0.41	0.05	0.40	0.05	0.40	0.10	0.42	0.05
SEC	TOP.5	0.41	0.19	0.50	0.05	0.53	0.09	0.47	0.07	0.47	0.10	0.48	0.08	0.48	0.09
	TOP.10	0.52	0.19	0.57	0.06	0.58	0.06	0.55	0.03	0.54	0.05	0.56	0.07	0.56	0.08
	TOP.15	0.56	0.20	0.58	0.06	0.60	0.06	0.58	0.04	0.59	0.07	0.60	0.07	0.60	0.08
	TOP.20	0.59	0.21	0.59	0.07	0.63	0.07	0.59	0.06	0.63	0.09	0.63	0.09	0.63	0.09
SPR	TOP.5	0.66	0.14	0.70	0.04	0.70	0.05	0.68	0.07	0.69	0.05	0.69	0.05	0.68	0.06
	TOP.10	0.74	0.12	0.75	0.05	0.75	0.05	0.76	0.07	0.76	0.04	0.74	0.05	0.76	0.05
	TOP.15	0.77	0.10	0.77	0.05	0.77	0.05	0.80	0.07	0.78	0.05	0.77	0.05	0.79	0.05
	TOP.20	0.80	0.10	0.78	0.05	0.79	0.05	0.81	0.06	0.80	0.05	0.78	0.05	0.80	0.06
WFly	TOP.5	0.58	0.09	0.47	0.17	0.53	0.15	0.60	0.06	0.50	0.25	0.38	0.21	0.60	0.06
	TOP.10	0.62	0.10	0.58	0.08	0.61	0.12	0.64	0.08	0.58	0.19	0.53	0.19	0.64	0.07
	TOP.15	0.64	0.12	0.66	0.09	0.66	0.09	0.66	0.08	0.60	0.19	0.60	0.17	0.66	0.07
	TOP.20	0.66	0.13	0.70	0.06	0.68	0.09	0.68	0.08	0.64	0.16	0.64	0.11	0.66	0.05

results comparable with the top meta-heuristic(s) in two cases out of 40 median values in Table 7.

5.1.1. Statistical analysis

Based on our preliminary analysis, *we would conclude that there is no "master" (dominant) meta-heuristic when configuring topic models for duplicate bug report identification.* In this section, we report the results of the statistical tests to verify whether such preliminary results hold from a statistical point of view.

According to the Friedman test, the differences among the different meta-heuristics are statistically significant independently of the fitness function being used. Indeed, we obtained a p-value $< 2.2 \times 10^{-16}$ for all three fitness functions. The ranks produced by the Friedman test and the Nemenyi test reveal that the top three meta-heuristics are SA, SHC, and GA, although they have different orders depending on which fitness function is used (see Fig. 1).

Fig. 1 plots the average ranks (in performance) achieved by each meta-heuristic across all ten projects in our benchmark. Meta-heuristics with lower average ranks are preferred as they perform better than those with larger rank. For the significance level $\alpha = 0.95$, the Figure also plots the critical difference CD as computed by the Nemenyi test (the grey area in the plots). The difference $d(A, B)$ between the average ranks of two meta-heuristics A and B is statistically significant if and only if it is larger than the critical distance, i.e., $d(A, B) > CD$.

According to ranks in Fig. 1, SHC is ranked first when using the *silhouette coefficient* and *topic coherence*, while it is ranked second when using the *raw score*. Vice versa, SA is ranked first when using the *raw score* and second when using either *silhouette coefficient* and *topic coherence*. GA is always ranked third independently of the fitness function being used. In terms of statistical significance, there is no statistically significant difference between the top three meta-heuristics when the *silhouette coefficient* and *topic coherence* according to the Nemenyi test. The relative distance between their ranks is lower than the critical distance $CD = 1.50$ determined by the test for the statistical significance. Instead, when using the *raw score*, SA and SHC are statistically better than all other meta-heuristics. This reveals that local solvers are preferable for this fitness function. Finally, CMA-ES, Random search, and DE are always ranked last, and their average ranks significantly differ (based on the critical distance $CD = 1.50$) from the top meta-heuristics.

> **Summary of RQ1**. *There is no "master" (dominant) meta-heuristic when configuring topic models for duplicate bug report identification. CMA-ES, Random search, and DE are less effective than the other meta-heuristics.*

5.2. Comparing the fitness functions

Fig. 2 depicts the distributions (box-plots) of the TOP_{20} scores achieved by the seven meta-heuristics when changing the fitness functions. In the box-plots, the results for the different functions are highlighted in different colors: red for *topic coherence*, green for the *raw score*, and blue for the *silhouette coefficient*. The results are further grouped by projects,

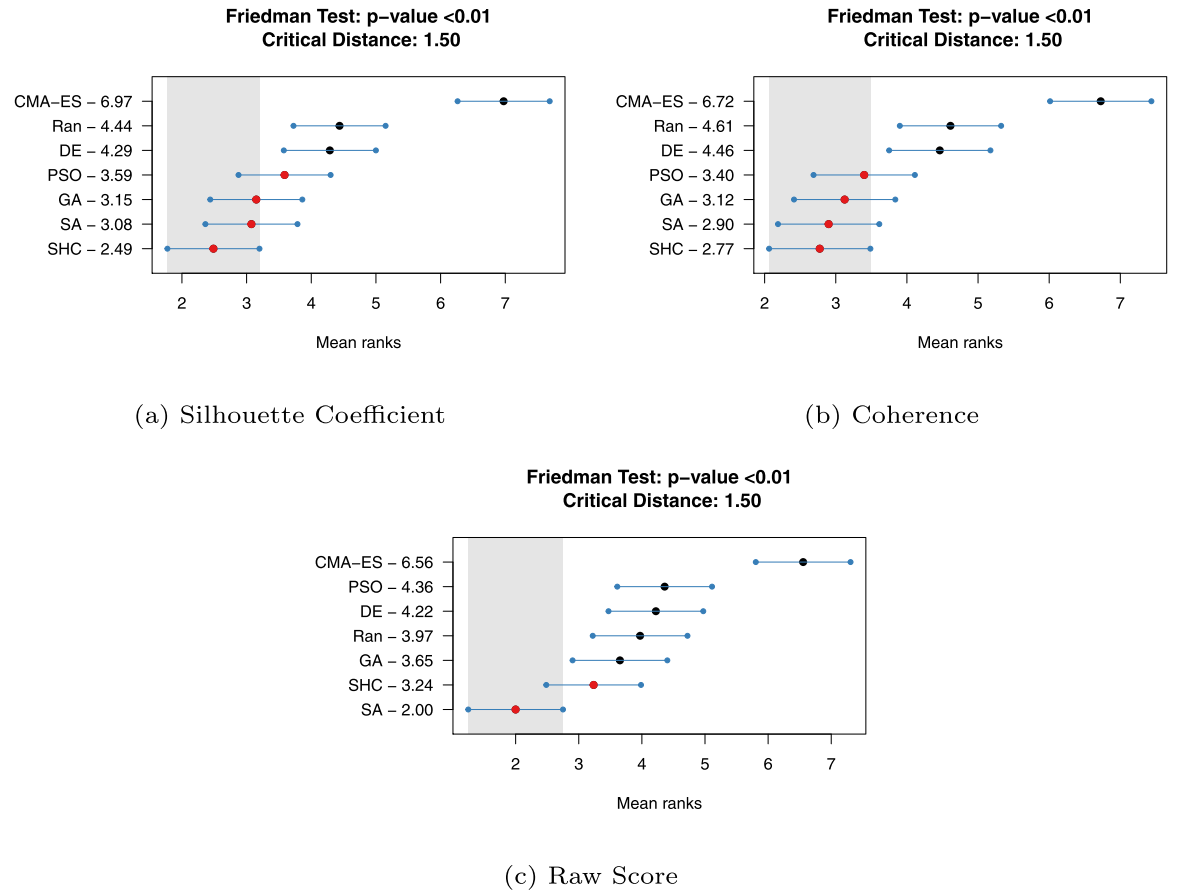


Fig. 1. Comparing the meta-heuristics with the Friedman test and the post-hoc Nemenyi test for each fitness function separately.

which correspond to the separated grey boxes in Fig. 2.

For `apache collections`, all three fitness functions are comparable as all meta-heuristics achieve the same optimal TOP_k scores. For `datacmns` and `sec`, *topic coherence* yields to higher TOP_{20} score for the majority of meta-heuristics. Instead, the *silhouette coefficient* achieves better results for the projects `hive`, `io`, `math`, and `spr`. For other projects, we obtained mixed results when choosing alternative meta-heuristics.

We also observe that CMA-ES performs better with the *raw score* than with other fitness functions in five out of 10 projects, namely `datacmns`, `hive`, `roo`, `sec`, and `wfly`. However, it leads to lower performances when combined with other solvers, such as GA (seven out of ten projects), PSO (five out of ten projects), and DE search (five out of seven projects).

Vice versa, we observe that the *silhouette coefficient* improves the performance of GA. Indeed, GA achieves better results with the *silhouette coefficient* than with the other fitness functions in five out of ten projects. It also improves the performance of other meta-heuristics, such as Random search (five out of ten projects) and DE (four out of ten projects).

To assess the statistical significance of the results, we rely on the Friedman test and the post-hoc Nemenyi test (see Fig. 3). The Friedman test reveals that there is no significant difference among the different fitness functions for DE (p-value $= 0.62$), GA (p-value $= 0.46$), random search (p-value $= 0.92$), and SA (p-value $= 0.34$). This means that the three different fitness functions produce equally good topics for these four meta-heuristics. Instead, we find significant difference for CMA-ES (p-value < 0.01), PSO (p-value < 0.01), and SHC (p-value < 0.01).

According to the post-hoc Nemenyi test, the *silhouette coefficient* is ranked first for six out of seven meta-heuristics, although the statistical significance holds only for two meta-heuristics (PSO and SHC). When using CMA-ES, the *raw score* achieves the best results (smaller rank); *topic coherence* is ranked second but with a rank distance lower than the critical distance $CD = 0.524$; finally, the *silhouette coefficient* is ranked third and it is significantly inferior to the other fitness functions. For PSO and SHC, we observe similar results: the *silhouette coefficient* is ranked first, followed by *topic coherence* and *raw score*. However, from a statistical point of view, the differences in the mean ranks between the *silhouette coefficient* and *topic coherence* are not statistically significant, while the *raw score* achieves a significantly worse rank.

> **Summary of RQ2.** *None of the evaluated fitness (surrogate) functions outperform the others when tuning LDA. However, we can make the following recommendations: use either silhouette coefficient or topic coherence for all meta-heuristics, except CMA-ES that works better with the raw score.*

5.3. Running time analysis

Table 8 reports the median (and the interquartile range) running time required by the evaluated search-based approaches (i.e., combinations of meta-heuristics and fitness functions) to reach the same stopping criterion (50 FEs) across 100 independent runs.

As we can observe, the running time diverges substantially among systems and fitness functions. Indeed, the running time depends on the size of the project: larger projects have corpora from which LDA extracts the topics. It also depends on the fitness functions, which compute different metrics and incur different overheads. For example, the *raw score* requires to re-run LDA ten times to compute the topic stability across independent runs. Instead, the *silhouette coefficient* and *topic coherence* do not require LDA re-runs. However, as explained in Section 4.6, these two fitness functions were combined with Gibb's sampling

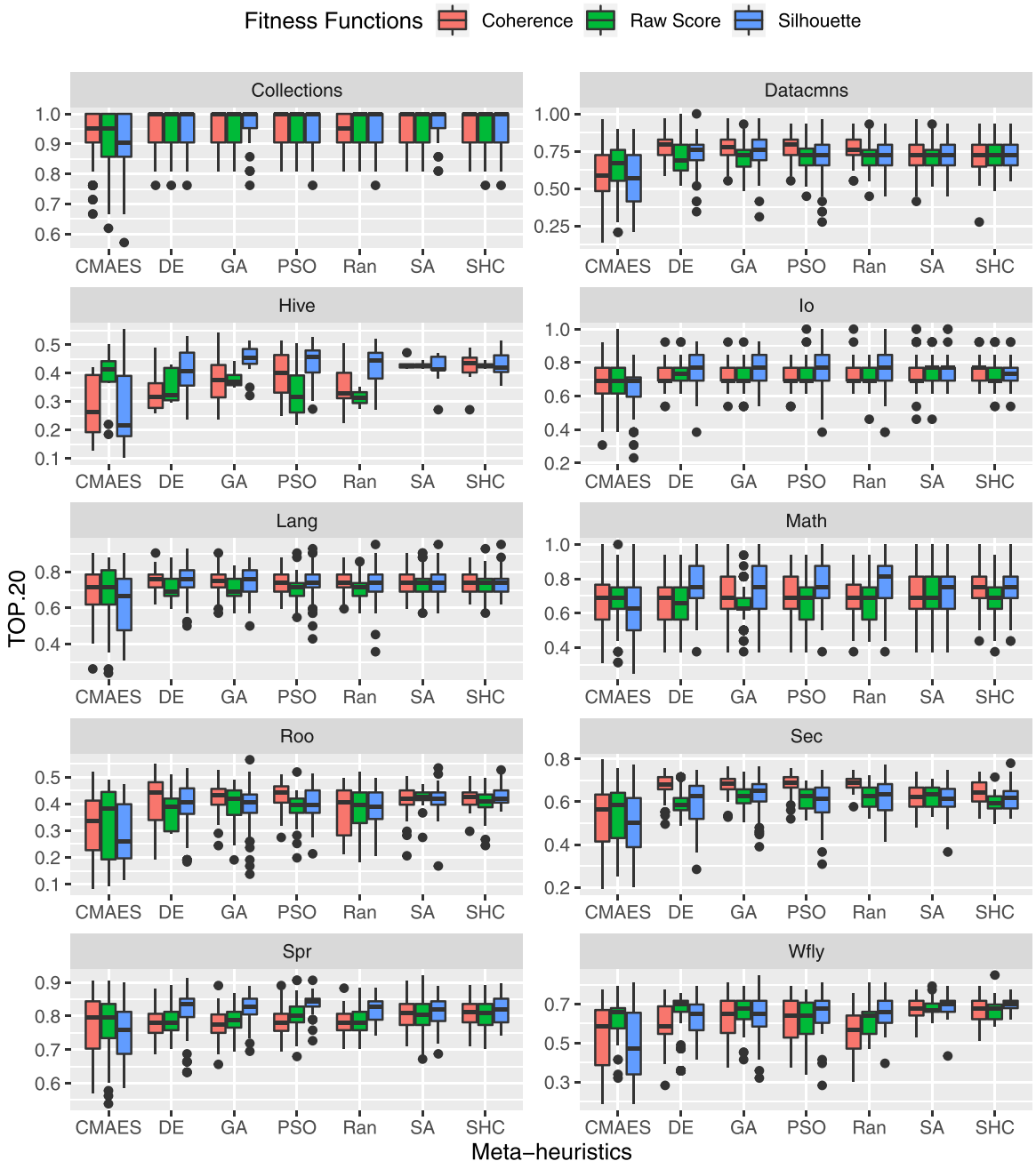


Fig. 2. TOP_{20} scores achieved by the meta-heuristics when varying the fitness function.

restarting strategy implemented in `topicmodels`. As a consequence, we observe that all meta-heuristics require more running time to perform 50 FEs when using the *raw score* compared to when using different fitness functions.

For what regards the meta-heuristics, we observe that GA and PSO incur the lowest search overhead in eight and two projects, respectively. The Friedman test revealed that the running time significantly differs among the different meta-heuristics (p-value $< 2.2 \times 10^{-16}$). Fig. 4(a) shows the ranking and the results of the post-hoc Nemenyi test. As we can observe, PSO, GA, and CMA-ES are the top three meta-heuristics in terms of running time. The differences in their average ranks are not significant because they are lower than the critical distance $CD = 1.64$ determined by the Nemenyi test. Instead, the other four meta-heuristics are significantly less performant (i.e., they require larger running time) than the top-three meta-heuristics.

Notice that our results diverge from what reported in previous studies. In particular, Agrawal et al [8] advocated for using DE as faster and more effective than GA. However, in their study, they used different stopping criteria for the two meta-heuristics, namely 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 tuning LDA, the difference in running time is not due to the internal complexity of the genetic and mutation operators implemented by the different meta-heuristics. The differences are due to the LDA configurations the meta-heuristics find. In particular, the running time to perform each FEs (e.g., running LDA with a given configuration) is proportional to the number of topics to extract. Therefore, combinations of meta-heuristics and fitness functions that generate a lower number of topics would also incur a lower search overhead.

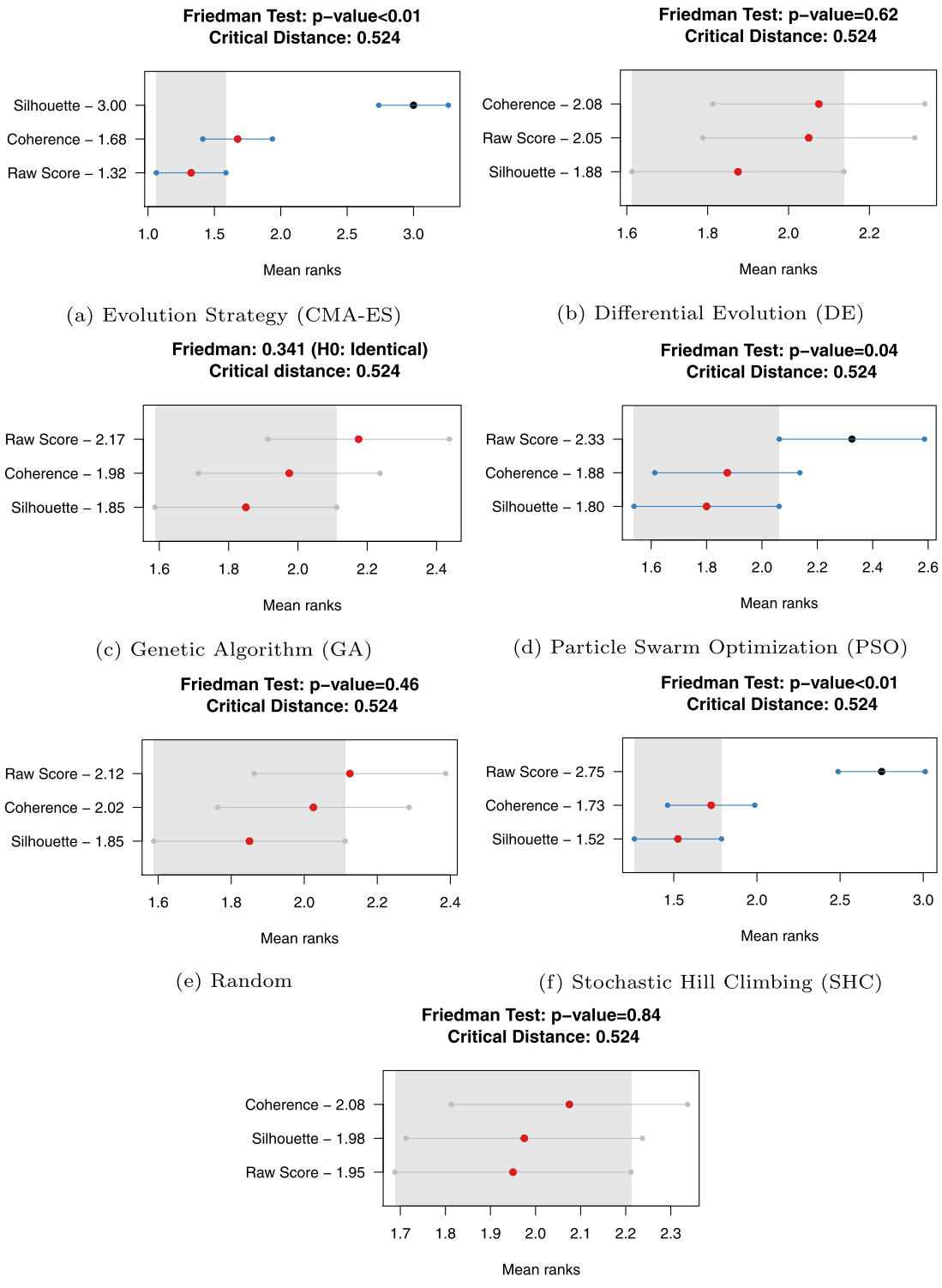


Fig. 3. Comparing the fitness functions with the Friedman test and the post-hoc Nemenyi test for each meta-heuristic separately.

To confirm such a hypothesis, we used a two-way permutation test [23] to verify whether there is a statistically significant interaction between the running time, the meta-heuristics, and the number of generated topics. The two-way permutation test is a non-parametric alternative to the two-way Analysis of Variance (ANOVA). As such, it does not require the data to be normally distributed and can be used for nominal values (i.e., the name of the meta-heuristics in our case) as well. This test applies an iterative procedure to compute the p-values using a different random permutation in each iteration. It is recommended to use the permutation test with a very large number of iterations to produce stable results [23]. We set the number of iterations to 10^8.

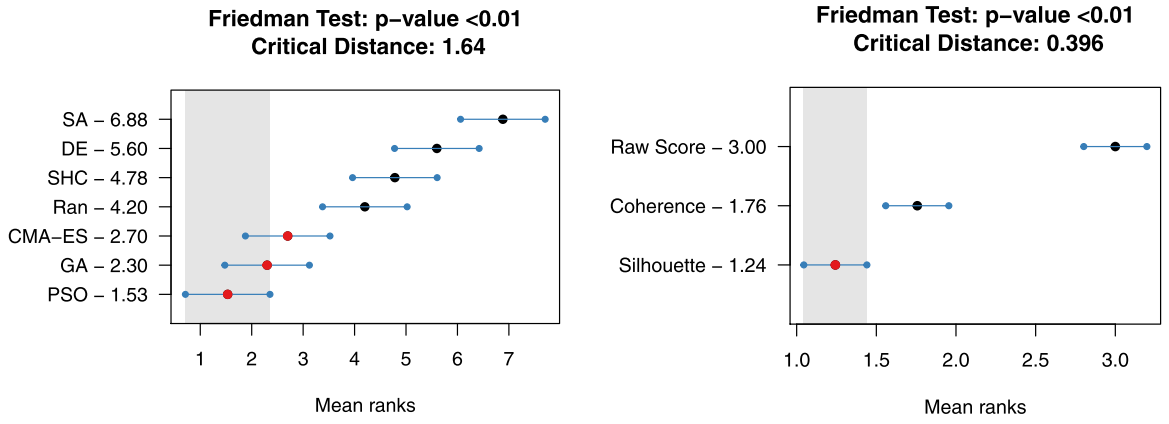
The two-way permutation test shows that the number of topics significantly interacts with the running time of the different search-based approaches (p-value $< 2.0 \times 10^{-16}$). The Spearman ρ

Table 8
Median and IQR of the running time (in minutes) required by the evaluated search-based approaches (i.e., pairs of meta-heuristics and fitness functions) to perform 50 fitness evaluations.

System	FF	CMA-ES		DE		GA		SHC		PSO		Ran		SA	
		Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR
Collections	Coh.	1.20	0.60	1.48	0.27	1.24	0.22	1.22	0.75	0.91	0.22	1.16	0.11	2.10	1.44
	R. Score	6.35	3.75	8.36	1.28	6.69	1.53	7.76	3.58	5.73	1.30	7.19	0.54	13.85	5.90
	S. Coeff.	0.98	0.70	1.25	0.22	0.78	0.18	1.13	0.84	0.77	0.17	1.12	0.08	1.78	1.42
Datacmns	Coh.	7.28	5.60	8.00	1.71	4.73	2.34	7.27	5.47	5.20	1.59	7.16	0.61	12.15	10.97
	R. Score	32.70	26.80	46.33	6.93	32.88	12.14	39.78	24.98	30.96	9.19	38.62	3.06	75.54	56.46
	S. Coeff.	7.43	5.51	8.13	1.52	4.54	2.02	6.48	5.54	4.97	1.16	6.92	0.61	11.08	9.97
Hive	Coh.	663.15	768.50	866.77	123.24	337.39	140.88	1031.83	582.01	692.46	253.78	871.95	97.06	1766.30	781.62
	R. Score	1242.32	1737.63	4348.48	1551.67	3302.83	2267.74	4398.21	1960.58	2489.55	435.36	5130.75	62.61	4398.21	1960.58
	S. Coeff.	750.47	785.49	1048.58	347.49	736.43	303.00	1191.54	670.30	638.58	166.38	852.95	104.48	1335.31	2227.59
Io	Coh.	1.29	0.80	1.70	0.26	1.37	0.26	1.18	0.78	1.07	0.33	1.39	0.11	2.22	1.73
	R. Score	6.34	4.97	9.85	1.21	7.41	1.62	9.03	5.24	6.43	1.80	8.27	0.68	15.30	8.52
	S. Coeff.	1.27	0.92	1.52	0.27	0.89	0.29	1.32	0.90	0.92	0.23	1.34	0.09	2.20	1.49
Lang	Coh.	7.34	6.10	9.54	2.01	7.23	1.89	9.02	6.44	6.09	1.62	8.24	0.90	12.97	12.31
	R. Score	31.89	38.42	53.94	10.60	45.27	10.74	49.72	28.39	33.93	8.74	44.51	3.20	83.42	68.36
	S. Coeff.	6.62	6.72	8.70	1.76	5.34	2.58	6.68	6.23	5.33	1.41	7.87	0.65	12.91	12.73
Math	Coh.	11.13	10.05	14.15	2.34	10.66	2.46	11.44	9.68	8.84	2.51	11.72	1.08	20.48	18.45
	R. Score	37.76	48.84	74.82	12.83	60.47	13.26	68.10	48.18	47.44	15.29	60.45	6.68	111.47	86.72
	S. Coeff.	11.12	9.87	12.75	2.95	7.22	3.55	10.18	9.18	7.93	2.00	10.99	1.21	18.32	16.47
Roo	Coh.	213.92	171.28	223.37	51.60	122.52	28.40	208.88	238.41	166.99	60.13	233.62	21.87	451.87	361.75
	R. Score	573.07	837.76	1202.64	564.65	818.18	354.27	1449.43	1482.15	1032.13	285.87	1403.35	130.79	1694.01	1643.07
	S. Coeff.	201.64	169.70	219.30	52.70	149.79	92.81	263.01	193.93	151.04	34.29	222.32	28.90	409.83	306.90
Sec	Coh.	86.51	66.82	90.88	21.75	50.33	11.50	102.01	99.01	69.85	19.47	91.88	10.68	145.63	117.47
	R. Score	279.07	295.00	570.60	152.86	435.77	174.53	561.93	446.45	355.11	109.98	485.48	84.42	647.49	511.19
	S. Coeff.	79.49	74.60	96.84	26.62	61.42	31.02	97.68	62.59	58.41	17.49	84.75	8.76	144.92	119.82
Spr	Coh.	5.90	3.62	7.58	1.35	6.36	1.24	6.74	3.41	4.59	1.31	6.07	0.60	10.80	8.38
	R. Score	26.76	17.52	40.81	7.05	33.03	5.78	32.07	23.48	24.72	5.77	32.93	2.95	57.18	28.17
	S. Coeff.	5.22	4.35	6.62	1.38	3.82	1.50	4.97	4.35	4.45	1.72	6.05	0.41	10.24	7.43
WFly	Coh.	580.20	639.24	695.13	162.15	276.66	102.34	695.29	618.03	540.26	182.20	757.90	75.52	1466.07	1463.26
	R. Score	2571.80	2544.03	4163.75	1674.63	3072.29	1230.25	4635.61	5800.84	3421.99	1085.59	4637.96	355.99	7267.03	3835.18
	S. Coeff.	565.87	647.17	851.58	159.42	490.41	275.11	764.73	837.07	488.90	130.70	720.63	55.43	1347.43	1043.48

Table 9
Comparing LDA with off-the-shelf parameter values versus LDA tuned with GAs and the silhouette coefficient.

System	off-the-shelf Parameters				Tuning with GA and Sil. Coeff.			
	TOP_5	TOP_{10}	TOP_{15}	TOP_{20}	TOP_5	TOP_{10}	TOP_{15}	TOP_{20}
Collection	0.81	0.86	0.90	0.90	0.95	1.00	1.00	1.00
Datacmns	0.31	0.45	0.58	0.62	0.38	0.62	0.72	0.76
Hive	0.21	0.21	0.28	0.34	0.54	0.69	0.77	0.77
Io	0.38	0.54	0.54	0.54	0.54	0.69	0.77	0.77
Lang	0.19	0.33	0.45	0.62	0.48	0.64	0.71	0.76
Math	0.25	0.56	0.62	0.75	0.44	0.62	0.75	0.75
Roo	0.11	0.12	0.15	0.18	0.24	0.34	0.38	0.40
Sec	0.15	0.28	0.40	0.49	0.48	0.58	0.63	0.65
Spr	0.43	0.64	0.75	0.81	0.61	0.76	0.80	0.83
WFly	0.21	0.21	0.28	0.34	0.53	0.59	0.64	0.65



(a) Results for meta-heuristics (b) Results for fitness functions

Fig. 4. Comparison of running time for meta-heuristics and fitness functions.

correlation coefficient indicates that the running time and the number of topics are strongly correlated (ρ=0.7981 and p-value $< 2.0 \times 10^{-16}$). Therefore, the larger the number of topics, the larger the running time. Instead, there is no statistical interaction between the meta-heuristics being used to tune LDA and the running time (p-value=0.38).

> **Summary of RQ3**. *The running time strongly depends on the number of topics selected by the meta-heuristics during the search. Instead, the internal complexity of the meta-heuristics is small or negligible.*

Our results also slightly diverge from our conference paper. In [19], we concluded that random search is slightly faster than our meta-heuristics. Instead, in this extension, we do not observe the same trend. However, this can be justified by a different configuration setting used for both GA and PSO. In particular, we replaced the *Gaussian mutation* (used in [19]) with the *polynomial mutation* (used in this article). We replaced the mutation operator because the *Gaussian mutation* can generate chromosomes with genes that do not satisfy the constraints (upper and lower bounds in our case). In our study, we observed that scenario was common for the hyperparameters $\alpha \in [0; 1]$ and $\beta \in [0; 1]$. Instead, the polynomial mutation always generates genes within these bounds.

From Table 8, we observe a clear trend in terms of running time for different fitness functions. The *silhouette coefficient* leads to lower running time overhead in 56 out of 70 times (10 datasets $\times$ 7 meta-heuristics). *Topic coherence* follows with the lowest running time in 14 out of 70 cases. Instead, the *raw score* is much heavier from a computational point of view. On average, the *raw score* is 5.7 times slower than the *silhouette coefficient*, and 5.2 times slower than *topic coherence*. These differences are also statistically significant according to the Friedman test (p-value $< 2.2 \times 10^{-16}$). Fig. 4(b) shows the results of the post-hoc Nemenyi test. We can notice that the test ranks the *silhouette coefficient* first. This means that this intrinsic metric incurs a lower overhead compared to the *raw score* and *topic coherence*. These differences are also significant as the differences between the mean ranks of the *silhouette coefficient*, and the two alternatives are larger than the critical distance $CD = 0.396$ determined by the Nemenyi test. The two-way permutation test further confirms these results as the choice of fitness function significantly interacts with the running time when tuning LDA (p-value $< 2.0 \times 10^{-6}$).

> **Summary of RQ4**. *The silhouette coefficient is significantly more efficient than the alternative fitness functions, namely topic coherence by Mimno et al [15]. and raw score by Agrawal et al [8].*

6. Discussions

In Section 5, we presented the empirical results to compare search-based approaches to tune LDA systematically. This section discusses tuned LDA vs. LDA performance configured with "off-the-shelf" parameter values in the context of duplicate bug reports identification. Section 6.2 provides some guidelines based on the quantitative results from Section 5. Finally, Section 6.3 discusses the connections between tuning LDA and parameter tuning for SBSE.

6.1. Does tuning LDA matter?

The importance of tuning LDA for software engineering tasks has already been investigated in the related literature. Panichella et al [9] showed a sound LDA calibration leads to substantially better performance compared to "off-the-shelf" (or "ad-hoc" configurations)

configurations for software engineering tasks. Panichella et al [9] reported +30% of the average precision (MAP) for traceability recovery, +32% of effective measure (EM) for feature location, and between +5% and +14% better overlap with topics manually-identified by developers for benchmark source code snippets.

Furthermore, Agrawal et al [8] showed that "off-the-shelf" settings lead to unstable results when generating topics for three different datasets: NASA soft- ware project and issue tracking system, StackOverflow, and CiteMap. More specifically, words and topics generated with untuned LDA change when altering the documents order over multiple runs (*order effects*). Therefore, *using standard LDA with its "off-the-shelf" settings should now be depreciated* [8].

The prior studies mentioned above provided empirical evidence to the need for tuning LDA for software related corpora. In this subsection, we investigate whether tuning LDA matters in the context of identifying duplicate bug reports. To this aim, we compare the performance (TOP_k scores) of LDA with "off-the-shelf" settings versus LDA tuned using GAs and the *silhouette coefficient*. As "off-the-shelf" setting, we use the following parameter values:

- The number of topics K=10, as done in [8]. This value also corresponds to the default setting in `mallet`[7].
- The number of Gibbs iterations N=300.
- $\alpha = 0.1$ and $\beta = 0.1$, which are widely used (default) hyperparameter values in the literature (e.g., [73–76]).
- LDA is a probabilistic model; as such, it can return slightly different results when running multiple times with different random seed. For the "off-the-shelf" configuration, we use random seed = 0, which is often used as the default seed in existing ML tools (e.g., in WEKA [77]).

Table 9 reports the results of the comparison. "Off-the-shelf" settings lead to lower TOP_k scores compared to tuned LDA. We can observe a large difference between "Off-the-shelf" settings and tuned LDA using the TOP_5 metric. The former retrieves on average (median value) 23% of the duplicated bug reports within the top five most similar past reports. When tuning the hyperparameters, LDA identifies on average (median value) 51% of duplicates by inspecting the top five most similar past reports. To be more concrete, let us consider the project `lang`; when using untuned LDA, developers would be able to identify only 19% of duplicate reports by inspecting the top five similar reports. Instead, when tuning LDA, developers would be able to identify 48% of the duplicates within the top five reports.

The differences between "Off-the-shelf" settings and tuned LDA remain consistent also for larger cut-offs, i.e., TOP_k scores with $K > 5$.

Our results confirm that tuning LDA is an important requirement to produce high-quality topics and achieve good results in SE tasks, including when identifying duplicate bug reports.

6.2. LDA Tuning guidelines

Our results in Section 5 suggest that there is no clear winner among different search-based approaches for tuning LDA. In fact, we observed that multiple meta-heuristics could achieve equally good performances (TOP_k scores). Besides, different fitness functions also lead to better results but in different projects and when combined with some meta-heuristics.

To derive general guidelines about which combinations of meta-heuristics and fitness functions to use, we rely on statistical analysis. In particular, we use the Friedman test with the post-hoc Nemenyi test. Fig. 5 shows the results of the statistical tests for the different combinations of meta-heuristics and fitness functions. The statistical tests reveal that the top 10 combinations are significantly better than the bottom 11 combinations ($p-$ value < 0.01). Based on the statistical results, we can draw the following guidelines:

G.1 We recommend using GAs with the *silhouette coefficient* for larger projects or when developers (or researchers) want to tune LDA in a shorter time. Our results indeed suggest that this combination produces LDA settings that yield TOP_k score equivalent to those achievable with SHC and SA. The results of RQ3 highlighted that GAs is less expensive than other meta-heuristics. Finally, the results for RQ4 indicate that the *silhouette coefficient* incurs a significantly lower overhead compared to the other fitness function.
G.2 For small/medium projects or when the running time is not a strong constraint, we recommend to use SHC or SA with either *silhouette coefficient* or *topic coherence*. These combinations of meta-heuristics and fitness functions achieve slightly better results (mean ranks in Fig. 5) but at the price of a larger overhead.
G.3 We do not recommend using CMA-ES, Random search, and DE as these meta-heuristics achieve significantly lower TOP_k scores than other meta-heuristics (see the mean ranks in Fig. 5).

6.3. Parameter tuning in search-Based software engineering

Tuning the parameters of the meta-heuristics has been widely investigated in the SBSE literature [78–80]. Parameter tuning has a "critical" impact of the performance of the meta-heuristics on given search problems or benchmark. However, Arcuri and Fraser [78,79] showed that the "default" parameter values suggested in the literature perform relatively well for test case generation. Besides, parameter tuning is acceptable for researchers, while practitioners have stronger time constraints. More recently, Kotelyanskii et al [80] performed a large-scale empirical study aimed at tuning EvoSuite [81] using the sequential parameter optimization toolbox (SPOT) [82]. Their results further confirmed prior findings: "default" parameter values for meta-heuristic search lead to results (i.e., structural coverage) comparable to those produced with tuning.

While there are similarities (i.e., parameter tuning) between our study and prior studies in SBSE, *there is a remarkable fundamental difference*. Prior studies [78–80] *focused on tuning meta-heuristics*. Instead, in this paper, *we tune LDA hyperparameters using meta-heuristics*. For the meta-heuristics, we use the default parameter values suggested in the related literature (see Section 4.3). *We do not tune the meta-heuristics* based on the following considerations:

- Meta-heuristics with "default" parameter values achieve "acceptable" results, as shown by Arcuri and Fraser [78,79], and Kotelyanskii et al [80].
- Tuning LDA is extremely expensive. Our study used very few fitness evaluations (i.e., 50) as a stopping condition. This results in an overhead that ranges from 46 seconds (for the smaller project in our study) up to eight hours (for the largest project) when using GAs and the *silhouette coefficient*. Running GAs (or other meta-heuristics) with different parameter values is therefore prohibitive.
- Even without tuning the meta-heuristics, LDA with tuned parameters produces better topics and achieves larger TOP_k scores compared to LDA with "off-the-shelf" parameter values (see Section 6.1). Therefore, our results confirm previous findings [8,9] as *using standard LDA with its "off-the-shelf" settings should now be deprecated* [8].

While the related literature focused on hyperparameter tuning for meta-heuristics, our results open new horizons for the SBSE community toward tuning machine learning (ML) and natural language processes (NLP) models. Even without tuning the meta-heuristics, search-based approaches have huge potentials to help researchers and practitioners automate the parameter tuning. Our study shows that without proper

[7] http://mallet.cs.umass.edu/topics.php

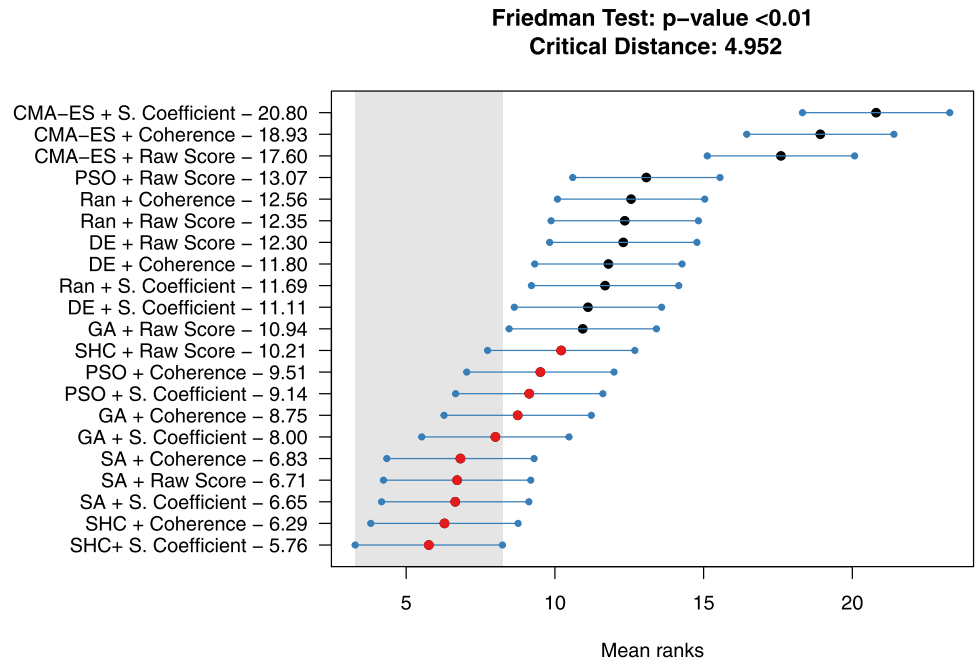


Fig. 5. Results of the Friedman test and the post-hoc Nemenyi test.

tuning, the performance (topic quality) of LDA is undermined, and search-based approaches can effectively solve the tuning problem. In the era of the massive application of ML to software engineering problems, we foresee SBSE applications to ML and NLP.

Future SBSE applications to tune ML and NLP should follow the three main steps we applied in this study:

1. Design/choose surrogate metrics for tuning ML in an unsupervised manner. We recommend unsupervised tuning because it does not require labeled data and addresses the potential threats to validity related to the dataset splitting in training and test sets.
2. Select alternative meta-heuristics from different categories of optimizers.
3. Benchmarking the different combinations of surrogate metrics and meta-heuristics considering two factors: the "optimality" of the produced parameters values and the tuning overhead.

7. 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. Besides, *raw score* re-run LDA multiple times with the same configuration to measure the topic stability across the runs.

The comparison among search-based approaches (i.e., meta-heuristics and fitness functions) is based on metrics (i.e., TOP_k) widely adopted in the literature to assess topic modeling methods when used to identify duplicate bug reports. The entire experiment was performed on the same dedicated server[8] to avoid bias when computing the running time.

Internal validity. We drew our conclusions by executing 100 independent runs to address the random nature of the evaluated search-based approaches. Besides, we use the Friedman tests, the post-hoc Nemenyi test, and the two-way permutation test to assess the statistical significance of the results.

External validity. In our study, we consider ten open-source projects from the `Bench4BL` dataset [6]. While we assessed search-based approaches to tune LDA and identify duplicate bug reports, further studies are needed to replicate our analysis in the context of other SE tasks.

8. Conclusion and future work

In this paper, we empirically compare different search-based approaches to tune LDA for software documents in an unsupervised manner. We consider the case of identifying duplicate bug reports in seven open-source projects from the `BenchBL` dataset [6]. Bug reports contain text and information written by external developers, and that can be used to debug and fix the code. Due to the textual nature of the reports, topic modeling and IR methods have been applied in the literature (e.g., [2,20]) to automate the process of retrieving report duplicates.

Our results show that *multiple meta-heuristics are comparable across different projects* and when optimizing different surrogate metrics. However, some combinations of meta-heuristics and surrogate metrics are more effective and efficient than others. In particular, we observe that GAs with the *silhouette coefficient* produce large (better) TOP_k score and with significant lower tuning overhead. The lower overhead is due to the lower number of selected topics rather than the internal complexity of the meta-heuristics. Besides, local solvers like SHC and SA are slightly more effective than GAs, although the differences are not statistically significant. Still, they incur a larger overhead due to the larger number of topics they select. Local solvers work very well with any surrogate metrics, and not only with the *silhouette coefficient*. Random search is not effective nor efficient in tuning/configuring LDA. Finally, tuning LDA helps retrieving twice as many duplicates bug reports than untuned LDA within the top five most similar past reports in the issue tracking systems.

While the SBSE community focused on hyperparameter tuning of meta-heuristics [78–80], our study shows that search-based approaches are very effective in tuning complex like LDA with very few fitness evaluations (samples). Our study opens new horizons for the SBSE community and further encourages researchers to focus on automated tuning of complex techniques, such as machine learning (ML) and natural language processing (NLP).

[8] The specifications of the server are detailed in Section 4.5.

Declaration of Competing Interest

None.

Acknowledgements

The author thanks the anonymous reviewers for their thorough feedback and invaluable suggestions.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.infsof.2020.106411.

References

[1] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, Information Retrieval models for recovering traceability links between code and documentation. The 16th IEEE International Conference on Software Maintenance, 2000, pp. 40–51.
[2] A.T. Nguyen, T.T. Nguyen, T.N. Nguyen, D. Lo, C. Sun, Duplicate bug report detection with a combination of Information Retrieval and Topic Modeling. The 27th IEEE/ACM International Conference on Automated Software Engineering, 2012, pp. 70–79.
[3] G. Sridhara, E. Hill, D. Muppaneni, L.L. Pollock, K. Vijay-Shanker, Towards automatically generating summary comments for Java methods. The 25th IEEE/ACM International Conference on Automated Software Engineering, ACM Press, 2010, pp. 43–52.
[4] S. Panichella, A. Panichella, M. Beller, A. Zaidman, H.C. Gall, The impact of test case summaries on bug fixing performance: An empirical investigation. 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), 2016, pp. 547–558.
[5] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, S. Panichella, Using IR methods for labeling source code artifacts: Is it worthwhile?. The 20th IEEE International Conference on Program Comprehension (ICPC), 2012, pp. 193–202.
[6] J. Lee, D. Kim, T.F. Bissyandé, W. Jung, Y. Le Traon, Bench4bl: reproducibility study on the performance of IR-based bug localization. The 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ACM, 2018, pp. 61–72.
[7] D.M. Blei, A.Y. Ng, M.I. Jordan, Latent dirichlet allocation, The Journal of Machine Learning Research 3 (2003) 993–1022.
[8] A. Agrawal, W. Fu, T. Menzies, What is wrong with topic modeling? and how to fix it using search-based software engineering, Inf. Softw. Technol. 98 (2018) 74–88.
[9] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, A. De Lucia, How to effectively use topic models for software engineering tasks? An approach based on genetic algorithms. The International Conference on Software Engineering, IEEE Press, 2013, pp. 522–531.
[10] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, S. Panichella, Labeling source code with information retrieval methods: an empirical study, Empirical Software Engineering 19 (5) (2014) 1383–1420.
[11] T.L. Griffiths, M. Steyvers, Finding scientific topics, Proc. of the National Academy of Sciences 101 (Suppl. 1) (2004) 5228–5235.
[12] Y. Teh, M. Jordan, M. Beal, D. Blei, Hierarchical dirichlet processes, J. Am. Stat. Assoc. 101 (476) (2006) 1566–1581.
[13] S. Grant, J.R. Cordy, Estimating the optimal number of latent concepts in source code analysis. The 10th International Working Conference on Source Code Analysis and Manipulation, 2010, pp. 65–74.
[14] R. Arun, V. Suresh, C.V. Madhavan, M.N. Murthy, On finding the natural number of topics with Latent Dirichlet Allocation: Some observations. Pacific-Asia conference on knowledge discovery and data mining, Springer, 2010, pp. 391–402.
[15] D. Mimno, H.M. Wallach, E. Talley, M. Leenders, A. McCallum, Optimizing semantic coherence in topic models. Proceedings of the conference on empirical methods in natural language processing, Association for Computational Linguistics, 2011, pp. 262–272.
[16] N. Aletras, M. Stevenson, Evaluating topic coherence using distributional semantics. Proceedings of the 10th International Conference on Computational Semantics (IWCS 2013)–Long Papers, 2013, pp. 13–22.
[17] T. Yarnguy, W. Kanarkard, Tuning latent dirichlet allocation parameters using ant colony optimization, Journal of Telecommunication, Electronic and Computer Engineering (JTEC) 10 (1–9) (2018) 21–24.
[18] A. Onan, Biomedical text categorization based on ensemble pruning and optimized topic modelling, Comput. Math. Methods Med. 2018 (2018).
[19] A. Panichella, A systematic comparison of search algorithms for topic modelling — A study on duplicate bug report identification. International Symposium on Search Based Software Engineering, Springer, 2019, pp. 11–26.
[20] A. Hindle, C. Onuczko, Preventing duplicate bug reports by continuously querying bug reports, Empirical Software Engineering 24 (2) (2019) 902–936.
[21] S. García, D. Molina, M. Lozano, F. Herrera, 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) (2009) 617–644.
[22] N. Japkowicz, M. Shah, Evaluating learning algorithms: A Classification perspective, Cambridge University Press, 2011.
[23] R.D. Baker, Modern permutation test software, in: E. Edgington (Ed.), Randomization Tests, Marcel Decker, 1995.
[24] H. Schütze, C.D. Manning, P. Raghavan, Introduction to information retrieval 39, Cambridge University Press Cambridge, 2008.
[25] E. Enslen, E. Hill, L.L. Pollock, K. Vijay-Shanker, Mining source code to automatically split identifiers for software analysis. The 6th International Working Conference on Mining Software Repositories, 2009, pp. 71–80.
[26] P. Willett, The porter stemming algorithm: then and now, Program 40 (3) (2006) 219–223.
[27] R. Baeza-Yates, B. Ribeiro-Neto, Modern information retrieval, Addison-Wesley, 1999.
[28] D. Binkley, D. Lawrie, Information retrieval applications in software maintenance and evolution, Encyclopedia of Software Engineering (2009).
[29] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, S. Panichella, On the role of the nouns in IR-based traceability recovery. The 17th IEEE International Conference on Program Comprehension, 2009.
[30] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, A. De Lucia, Parameterizing and assembling IR-based solutions for SE tasks using genetic algorithms. 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER) 1, IEEE, 2016, pp. 314–325.
[31] G. Antoniol, G. Canfora, A. De Lucia, E. Merlo, Recovering code to documentation links in OO systems. Proc. of 6th Working Conference on Reverse Engineering, IEEE CS Press, Atlanta, Georgia, USA, 1999, pp. 136–144.
[32] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, E. Merlo, Recovering traceability links between code and documentation, IEEE Trans. Software Eng. 28 (10) (2002) 970–983.
[33] A. Panichella, C. McMillan, E. Moritz, D. Palmieri, R. Oliveto, D. Poshyvanyk, A. De Lucia, When and how using structural information to improve IR-based traceability recovery. 2013 17th European Conference on Software Maintenance and Reengineering, IEEE, 2013, pp. 199–208.
[34] G. Capobianco, A.D. Lucia, R. Oliveto, A. Panichella, S. Panichella, Improving IR-based traceability recovery via noun-based indexing of software artifacts, Journal of Software: Evolution and Process 25 (7) (2013) 743–762.
[35] N. Bettenburg, R. Premraj, T. Zimmermann, S. Kim, Duplicate bug reports considered harmful... really?. 2008 IEEE International Conference on Software Maintenance IEEE, 2008, pp. 337–345.
[36] J. Anvik, L. Hiew, G.C. Murphy, Coping with an open bug repository. Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange, ACM, 2005, pp. 35–39.
[37] J. Anvik. Assisting bug report triage through recommendation, University of British Columbia, 2007. Ph.D. thesis.
[38] C. Sun, D. Lo, X. Wang, J. Jiang, S.-C. Khoo, A discriminative model approach for accurate duplicate bug report retrieval. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1, ACM, 2010, pp. 45–54.
[39] Y. Tian, C. Sun, D. Lo, Improved duplicate bug report identification. 2012 16th European Conference on Software Maintenance and Reengineering, IEEE, 2012, pp. 385–390.
[40] T. Minka, J. Lafferty, Expectation-propagation for the generative aspect model. The 18th conference on Uncertainty in artificial intelligence, Morgan Kaufmann Publishers Inc., 2002, pp. 352–359.
[41] X. Wei, W.B. Croft, LDA-based document models for ad-hoc retrieval. The 29th Annual International Conference on Research and Development in Information Retrieval, ACM, 2006, pp. 178–185.
[42] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, M. Welling, Fast collapsed Gibbs sampling for Latent Dirichlet Allocation. The 14th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2008, pp. 569–577.
[43] C. Bird, T. Menzies, T. Zimmermann, The art and science of analyzing software data, Elsevier, 2015.
[44] M. Hughes, D.I. Kim, E. Sudderth, Reliable and scalable variational inference for the hierarchical Dirichlet process. Artificial Intelligence and Statistics, 2015, pp. 370–378.
[45] D. Binkley, D. Heinz, D. Lawrie, J. Overfelt, Source code analysis with LDA, Journal of Software: Evolution and Process 28 (10) (2016) 893–920.
[46] M.V. Mantyla, M. Claes, U. Farooq, Measuring LDA topic stability from clusters of replicated runs. The 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, 2018, p. 49.
[47] K. Stevens, P. Kegelmeyer, D. Andrzejewski, D. Buttler, Exploring topic coherence over many models and many topics. The 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, Association for Computational Linguistics, 2012, pp. 952–961.
[48] G.A. Miller, The magical number seven, plus or minus two: some limits on our capacity for processing information. Psychol. Rev. 63 (2) (1956) 81.
[49] J. Campos, Y. Ge, N. Albunian, G. Fraser, M. Eler, A. Arcuri, An empirical evaluation of evolutionary algorithms for unit test suite generation, Inf. Softw. Technol. 104 (2018) 207–235.
[50] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, Journal of Machine Learning Research 13 (2) (2012) 281–305.
[51] M. Mitchell, J.H. Holland, S. Forrest, When will a genetic algorithm outperform hill climbing. Advances in neural information processing systems, 1994, pp. 51–58.
[52] S. Sivanandam, S. Deepa, Introduction to genetic algorithms, Springer Berlin Heidelberg, 2007.
[53] P.J. Van Laarhoven, E.H. Aarts, Simulated Annealing. Simulated annealing: Theory and applications, Springer, 1987, pp. 7–15.

[54] K. Deb, D. Deb, Analysing mutation schemes for real-parameter genetic algorithms, International Journal of Artificial Intelligence and Soft Computing 4 (1) (2014) 1–28.
[55] D. Whitley, A.M. Sutton, Genetic Algorithms-a Survey of Models and Methods. Handbook of natural computing, Springer Berlin Heidelberg, 2012, pp. 637–671.
[56] S. Das, P.N. Suganthan, Differential evolution: a survey of the state-of-the-art, IEEE Trans. Evol. Comput. 15 (1) (2010) 4–31.
[57] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory. The 6th Intern. Symposium on Micro Machine and Human Science, 1995, pp. 39–43.
[58] Y. Zhang, S. Wang, G. Ji, A comprehensive survey on particle swarm optimization algorithm and its applications, Mathematical Problems in Engineering 2015 (2015).
[59] D.E. Gbenga, E.I. Ramlan, Understanding the limitations of particle swarm algorithm for dynamic optimization tasks: a survey towards the singularity of PSO for swarm robotic applications, ACM Computing Surveys (CSUR) 49 (1) (2016) 1–25.
[60] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evol. Comput. 9 (2) (2001) 159–195.
[61] S. Kumar, R. Naresh, Efficient real-coded genetic algorithm to solve the non-convex hydrothermal scheduling problem, International Journal of Electrical Power & Energy Systems 29 (10) (2007) 738–747.
[62] K. Price, R.M. Storn, J.A. Lampinen, Differential evolution: A practical approach to global optimization, Springer Science & Business Media, 2006.
[63] Y.-L. Zheng, L.-H. Ma, L.-Y. Zhang, J.-X. Qian, On the convergence analysis and parameter selection in particle swarm optimization. Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 03EX693) 3, IEEE, 2003, pp. 1802–1807.
[64] K. Deb, R.B. Agrawal, et al., Simulated binary crossover for continuous search space, Complex systems 9 (2) (1995) 115–148.
[65] B. Grün, K. Hornik, Topicmodels: an r package for fitting topic models, J. Stat. Softw. 40 (13) (2011) 1–30.
[66] L. Scrucca, GA: A package for genetic algorithms in r, Journal of Statistical Software, Articles 53 (4) (2013) 1–37.
[67] K. Mullen, D. Ardia, D. Gil, D. Windover, J. Cline, Deoptim: an r package for global optimization by differential evolution, J. Stat. Softw. 40 (6) (2011) 1–26.
[68] J. Richter, randomsearch: Random Search for Expensive Functions, 2019.
[69] D.M. Manfred Gilli, E. Schumann, Numerical Methods and Optimization in Finance (NMOF), 2011.
[70] A. Ghalanos, B. Pfaff, M.S. Lobo, L. Vandenberghe, S. Boyd, H. Lebret, Package 'parma', 2014.
[71] A. Sureka, P. Jalote, Detecting duplicate bug report using character n-gram-based features, in: 2010 Asia Pacific Software Engineering Conference, IEEE, pp. 366–374.
[72] D. Valcarce, A. Bellogín, J. Parapar, P. Castells, On the robustness and discriminative power of information retrieval metrics for top-n recommendation. Proceedings of the 12th ACM Conference on Recommender Systems, 2018, pp. 260–268.
[73] B. Subeno, R. Kusumaningrum, et al., Optimisation towards latent dirichlet allocation: its topic number and collapsed gibbs sampling inference process. International Journal of Electrical & Computer Engineering (2088–8708) 8 (2018).
[74] S. Fukuda, Y. Tomiura, E. Ishita, Research paper search using a topic-based boolean query search and a general query-based ranking model. International Conference on Database and Expert Systems Applications, Springer, 2019, pp. 65–75.
[75] T. Yoshida, R. Hisano, T. Ohnishi, Gaussian hierarchical latent dirichlet allocation: bringing polysemy back, arXiv preprint arXiv:2002.10855 (2020).
[76] Y. Zhang, M. Chen, D. Huang, D. Wu, Y. Li, Idoctor: personalized and professionalized medical recommendations based on hybrid matrix factorization, Future Generation Computer Systems 66 (2017) 30–35.
[77] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA data mining software: an update, ACM SIGKDD explorations newsletter 11 (1) (2009) 10–18.
[78] A. Arcuri, G. Fraser, On parameter tuning in search-based software engineering. International Symposium on Search Based Software Engineering, Springer, 2011, pp. 33–47.
[79] A. Arcuri, G. Fraser, Parameter tuning or default values? an empirical investigation in search-based software engineering, Empirical Software Engineering 18 (3) (2013) 594–623.
[80] A. Kotelyanskii, G.M. Kapfhammer, Parameter tuning for search-based test-data generation revisited: Support for previous results. 2014 14th International Conference on Quality Software, IEEE, 2014, pp. 79–84.
[81] G. Fraser, A. Arcuri, EvoSuite: automatic test suite generation for object-oriented software. Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, 2011, pp. 416–419.
[82] T. Bartz-Beielstein, SPOT: An r package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization, arXiv preprint arXiv:1006.4645 (2010).