Ranking approaches for similarity-based web element location

The Journal of Systems and Software 222 (2025) 112286

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

Ranking approaches for similarity-based web element location[☆]

Riccardo Coppola[a,*], Robert Feldt[b], Michel Nass[c], Emil Alégroth[c]

[a] Politecnico di Torino, Turin, Italy
[b] Chalmers University of Technology, Göteborg, Sweden
[c] Blekinge Institute of Technology, Karlskrona, Sweden

ARTICLE INFO

Keywords:
GUI testing
Test automation
Test case robustness
Web element locators
XPath locators

ABSTRACT

Context: GUI-based tests for web applications are frequently broken by fragility, i.e. regression tests fail due to changing properties of the web elements. The most influential factor for fragility are the locators used in the scripts, i.e. the means of identifying the elements of the GUI.
Objective: We extend a state-of-the-art Multi-Locator solution that considers 14 locators from the DOM model of a web application, and identifies overlapping nodes in the DOM tree (VON-Similo). We augment the approach with standard Machine Learning and Learning to Rank (LTR) approaches to aid the location of web elements.
Method: We document an experiment with a ground truth of 1163 web element pairs, taken from different releases of 40 web applications, to compare the robustness of the algorithms to locator weight change, and the performance of LTR approaches in terms of MeanRank and PctAtN.
Results: Using LTR algorithms, we obtain a maximum probability of finding the correct target at the first position of 88.4% (lowest 82.57%), and among the first three positions of 94.79% (lowest 91.86%). The best mean rank of the correct candidate is 1.57.
Conclusion: The similarity-based approach proved to be highly dependable in the context of web application testing, where a low percentage of matching errors can still be accepted.

1. Introduction

Testing is essential in ensuring software quality, but it is typically very time-consuming and therefore costly (Grechanik et al., 2009c,a). Despite the challenges, many reports emphasize the efficiency and cost-effectiveness of test automation in contributing to delivering high-quality software releases (Olan, 2003; Adamoli et al., 2011; Alegroth et al., 2013). Automated regression testing is a widely adopted approach in software development to evaluate and ensure the quality of each release. This method, when applied at the Graphical User Interface (GUI) level, involves creating a suite of test scripts that emulate user scenarios and verify the software's behavior using oracles (Liebel et al., 2013; Mahmud et al., 2014). However, new software releases often include changes that can disrupt automated regression tests, leading to the need for test maintenance. Maintenance is, therefore, a continuous activity that requires additional effort and expenses to keep the test suite working and relevant, over time. For GUI tests, these maintenance costs are exceptionally high since GUI structure, components, and logic frequently change with each release (Tonella et al., 2014; Alégroth and Feldt, 2017; Dobslaw et al., 2019; Nass et al., 2021). GUI level regression tests are thereby affected by changes to the GUIs appearance as well as the underlying logic and structure. The design of GUIs, intended for human use, creates additional difficulties for automation. Issues related to the synchronization between the test scripts and the application under test, which are less common in lower-level testing methods such as unit testing, add to these challenges (Olan, 2003; Nass et al., 2021).

Automated testing of GUI applications can be performed using various methods, but the most common for web apps is to base it on information from the Document Object Model (DOM) (Online, 2022b). A similar approach can also apply to desktop GUI testing, where meta-information about GUI elements is typically accessible through the OS or the specific GUI library being used. In DOM-based testing, GUI web elements used in tests, such as buttons, text fields, labels, etc., are located using DOM properties like element attributes, element text, IDs, XPaths (Online, 2022c), and CSS selectors (Online, 2022a).

☆ Editor: Dr. Dario Di Nucci.
* Corresponding author.
E-mail addresses: riccardo.coppola@polito.it (R. Coppola), robert.feldt@chalmers.se (R. Feldt), michel.nass@bth.se (M. Nass), emil.alegroth@bth.se (E. Alégroth).

https://doi.org/10.1016/j.jss.2024.112286
Received 23 April 2024; Received in revised form 11 November 2024; Accepted 15 November 2024
Available online 3 December 2024

However, DOM properties are prone to change, impacting the stability of automated test execution as the application evolves. This sensitivity is known as "test script fragility", manifesting as test failures with false positive – Tests incorrectly reporting defective behaviors – test results, and frequently causes increased maintenance, cost, and decreased quality, as reported by researchers (Memon et al., 2001; Grechanik et al., 2009b; Alegroth et al., 2015; Alégroth et al., 2018; Mahmud et al., 2014; Moreira et al., 2017; Coppola et al., 2018, 2019). Of course, significant changes to a SUT should result in a test failure, as it may indicate a defect. However, changes that unintentionally lead to test failures, even with minor modifications to the website, can break test execution, even though a manual tester would consider the test successful. These minor test failures lead to unnecessary debugging and maintenance, especially when the cause of failure is small and hard to detect. In the literature, unpredictable test scripts are often referred to as fragile or lacking robustness (Eladawy et al., 2018).

Research has had limited success in addressing the challenge of robust GUI element localization (Nass et al., 2021). Some studies have explored new locator methods such as image recognition (Yeh et al., 2009) or multi-locators (Leotta et al., 2016). However, despite these efforts, web element localization remains an unresolved challenge, and more research is needed to enhance the robustness and maintainability of GUI testing techniques and tools (Nass et al., 2021).

In our previous works, we have proposed a novel approach to web element localization for web applications, with a locator algorithm called Similarity-based web element localization (Similo) (Nass et al., 2022). The primary purpose of Similo is to increase the robustness of locating web elements by comparing the similarity of multiple web element attributes to achieve high stability and robustness in the execution of GUI-based tests over time. Similo takes advantage of information from multiple attributes by considering a weighted sum of the results of the comparisons of individual attributes of the web elements. To improve the approach further, the Similo algorithm has also been extended with a concept referred to as Visual Overlapping Nodes (VON), which was designed to extend the set of matching web elements for each target by leveraging the common structure of web applications (Nass et al., 2023a). The benefits of the VON approaches (in terms of localization precision and accuracy) have been validated on an experimental sample of web applications.

In this work, we conceptualize and empirically evaluate a new locator finding strategy, based on learning to rank algorithms, which we deem beneficial towards full automation of GUI testing activities. The core difference from related works is that previous solutions evaluated the locator problem as a binary classification task (using a similarity threshold for matching web elements), i.e., either finding or failing to find the best candidate matching the target. Failed localization's thereby results in prematurely terminated test execution or necessary human intervention. Even though existing approaches do provide ranked lists of candidates for a given target, no ML techniques are used to optimize the outcome of the ranking. Since the so called Learning-to-Rank (LtR) algorithms directly targets the optimization of ML models that can rank a large set of items, they are a natural choice to evaluate for the ranking of web elements. While there are many potential search and/or optimization methods that could be used to rank items we note that LtR algorithms have previously been investigated in software engineering, e.g. for ranking defect proneness of modules (Yu et al., 2023), and that some of the standard approaches that is based on random forests, showed promising results. Here, we thus focus on such classic LtR approaches and what they can provide in our problem setting.

The main contributions of the present work are the following:

- We introduce Machine Learning (ML) and Learning to Rank (LTR) approaches for web element localization, building upon our previous contributions, including the SIMILO algorithm (Nass et al., 2022) and the Visually Overlapping Nodes (VON) concept (Nass et al., 2023a). We then compare the ML and LTR approaches with the state-of-the-art and evaluate the benefits that can be obtained with such techniques to reduce test execution fragility;
- We analyze the benefits of the Ranking-based approach to locator identification in both the original SIMILO and VON-SIMILO setting.
- We finally discuss the benefits and drawbacks of such an approach and its feasibility in a real-world testing scenario.

The results that we collected when evaluating Learning to Rank algorithms to similarity-based locators for web application testing hold promise for an application of the techniques in state-of-the-art automated layout-based web application testing tools. The results we document in the present manuscript demonstrate that LTR approaches can be effectively applied to solve the task of correctly ranking correspondent web elements in the layouts of web pages that evolve over time, with a peak ranking performance of 88.45%.

This paper is structured as follows. Section 2 presents related work, and gives a background of web element locators and the VON Similo approach. Section 3 covers the details of the proposed Similo algorithm. The design, research questions, and procedure of the empirical study are presented in Section 4, and the results in Section 5. In Section 6, we discuss the results and sample use cases to motivate the use of the compared techniques. We state conclusions and future work in Section 8.

2. Background and related work

This section provides background information about automated web application GUI testing, state-of-the-art multi-locator approaches for web testing, and Learning To Rank algorithms along with their utilization in the Software Engineering literature.

2.1. Property-based GUI testing and test fragility

GUI testing is a software testing technique that involves executing test cases – either manually or automatically – against the graphical user interface (GUI) of an application to ensure that it is working as intended.

Alégroth et al. proposed a classification of GUI testing techniques in *generations*, according to the nature of the locators they use (Alégroth et al., 2015): first generation, or coordinate-based, techniques use exact coordinates on the screen to identify web elements; second generation, or property-based, techniques leverage the values of the web element attributes defined in the DOM of the webpage (e.g., for web applications: ids, text content, content descriptions, xpaths); third generation, or visual, techniques leverage the exact graphical appearance of the web elements to identify and interact with them. First-generation techniques have been largely abandoned due to the inherent fragility of using exact coordinates; at the same time, third-generation techniques have been shown to be fragile to change (Coppola et al., 2021). Most research and practice, thereby, has been, and still is, focused on property-based web application GUI testing.

In the web domain, a web element *locator* is defined as a method, function, approach, or algorithm that can locate a web element in a given web page according to a specific parameter. State-of-the-art property-based techniques and tools typically make use of conditions on the attributes of the web elements in the HTML DOM tree. Popular testing tools (e.g., Selenium) provide the possibility to use locators such as XPath or CSS expressions to locate elements on the web page. We refer to these types of locators as *single-locators*.

Single-locators are a reported source of *fragility* for GUI tests. Fragility is defined as the lack of robustness of the locators to identify a web element after changes in the GUI definition of the SUT. Several studies in the literature have identified single-locators as a common source of fragility since the attributes of web elements are often changed during updates of an application.

Fragility leads to false positive test results, where test failures are not caused by actual defects in the SUT, but by failing locators in the test code itself (Coppola et al., 2018).

When a locator cannot be found in the DOM tree of the current web page, after the SUT is updated, it is typically referred to as a *Broken locator* (Leotta et al., 2015).

Furthermore, the test fragility issue is increasing with the growing complexity of web applications and adoption of new frameworks, as well as the use of dynamic generations of attributes of web elements (Ricca et al., 2019). As a consequence, the demand for test case maintenance has increased.

Web application testing literature has addressed the fragility problem with different approaches. For instance, some efforts in literature have proposed repair strategies to automatically fix broken locators . Other solutions have tried to mitigate the issue by increasing the number of attributes involved in web element location, therefore providing multiple possibilities for how to identify the same web element in case some attributes are no longer able to identify it. Leotta et al. proposed a new type of web element locator, named *multi-locator*, which selects the best locator among a candidate set of locators produced by different algorithms (e.g., using XPaths, IDs, or text content); this selection is based on a voting procedure that assigns different weights to the different locator generation algorithms (Leotta et al., 2015).

Montoto et al. proposed a single-locator algorithm, which generates multiple XPath expressions to use as locators through a bottom-up iterative strategy; the algorithm starts from simple XPath expressions and concatenates sub-expressions until it is able to identify the desired target elements (Montoto et al., 2011). The algorithm works in the same fashion as a multi-locator approach since the XPath is initially composed by taking into account the textual content of the sought web element; then, if the XPath is not unique, the ancestors of the web element and their attribute values are used, iteratively, until the root is reached.

Other single-locator algorithms for generating robust XPath's are ROBULA (Leotta et al., 2014) and ROBULA+ (Leotta et al., 2016), proposed by Leotta et al.

The ROBULA+ algorithm (Leotta et al., 2016) generates generic XPath locators selecting all the nodes in the DOM tree, then iteratively applies transformations to generate specialized XPath expressions, selecting only the element of interest for a test sequence. The implementation of the tool employs several heuristic prioritization and black listing techniques for the selection of the XPath corresponding to the web elements of interest and collects multiple data points for each element to be stored and used in XPaths, such as tag names and attributes.

2.2. Learning to rank in software engineering

Learning to Rank (LtR), sometimes also known as Machine-Learned ranking (MLR), is a general approach in Machine Learning (ML) with the goal to rank a set of items *as a whole* (Liu et al., 2009). This is, in contrast to simpler and earlier approaches in ML, where such ranking is a side effect of predicting values/scores per individual item and then post-sorting the items to get a ranking. The canonical application example is in information retrieval, where a large list of documents needs to be ranked based on their relevance for a search query.

In modern LtR solutions, there are three main approaches of how the ranking is actually achieved (Cao et al., 2007; Liu et al., 2009). In the *pointwise approach*, a single value is predicted per individual item and then used to sort items from high to low (or low to high, depending on setup). The problem is thus seen as a regression problem where a ranking score is the output to be predicted. Almost any ML algorithm can be used for this purpose in the LtR setting. In the *pairwise approach*, the machine learning (ML) model takes information about two items as input and then indicates which one should be ranked higher. By calling the learning function multiple times, one can then produce (at least) a partial order of all the items. Finally, in the *listwise approach* the ML model directly outputs the ranked list by trying to optimize one of the LtR-specific loss functions (that can take the full list into account). Even though early empirical studies indicated that listwise approaches performed better, recent studies have shown that even traditional, pointwise approaches can be competitive and sometimes more robust (Ibrahim, 2020).

While any traditional ML algorithm can be used in the LtR setting, studies typically use either Random Forests, Gradient Boosting Machines, or Neural Networks. The main difference lies in the loss function used while training the models. For example, while traditional ML and the pointwise LtR setting typically use traditional accuracy-based loss functions such as mean root square error, the pairwise setting uses variants of classification accuracy, and listwise LtR approaches directly try to quantify how close to the top the most relevant items are ranked. They thus typically all use loss functions like Normalized Discounted Cumulative Gain (NDCG) (Järvelin and Kekäläinen, 2017) or Expected Reciprocal Rank (ERR) (Chapelle et al., 2009) which can reflect multiple levels of relevance.

Learning to rank algorithms has been applied to several Software Engineering problems. Safdari et al. use the history of previously localized bugs and their dependencies as features to rank files in terms of their likelihood of being the root cause of a bug (Safdari et al., 2019). Other studies have used the approach for bug triaging to improve the efficiency of bug assignee recommendation based on similar issues fixed previously, Tian et al. (2016) or to predict defect-prone software modules (Yang et al., 2014).

Projects receive many bug reports, and resolving these reports takes considerable time and human resources. To aid developers in the resolution of bug reports, various automated techniques have been proposed to identify and recommend developers to address newly reported bugs. Two families of bug assignee recommendation techniques include those that recommend developers who have fixed similar bugs before (a.k.a. activity-based techniques), and those that recommend suitable developers based on the location of the bug (a.k.a. location-based techniques). Previously, each of these techniques has been investigated separately. In this work, we propose a unified model that combines information from both developers' previous activities and suspicious program locations associated with a bug report in the form of similarity features. We have evaluated our proposed approach on more than 11,000 bug reports from Eclipse JDT, Eclipse SWT and ArgoUML projects. Our experiments show that our unified model can outperform a location-based baseline by Anvik et al. and an activity-based baseline by Shokripour et al. In terms of correct recommendations at top-1 position, our unified model outperforms the activity-based baseline 50.0

A number of recent studies applied LtR algorithms to test case prioritization. Bertolino et al. combined it with reinforcement learning for test selection and prioritization in Continuous Integration. Their approach automatically extracts code changes and test metrics to prioritize the features to be tested by obtaining a ranked test list (Bertolino et al., 2020). ML-based test case prioritization in the context of CI builds has been explored also by Yaraghi et al. (2022). A similar approach was also evaluated in an industrial case study (Omri and Sinz, 2022).

3. A data-driven ranking perspective for web element selection

In this section, we outline the different components of the Similo approach (Nass et al., 2023a), describing both the original Similo algorithm as well as the extension VON Similo.owever, we note that even though they are both based on calculating the similarity score between a target web element and a large set of candidate web elements, their conceptualizations and empirical evaluations views them as *binary classification models*; they either match the correct target element or not. The main argument of this paper is rather that they can be seen as *ranking models*, returning a whole list of candidate elements, ordered by similarity to the target. Also, by tuning any such model in a data-driven manner it should be possible to further improve performance.

From this new perspective, i.e. *web element selection as data-driven ranking*, it is natural to consider machine learning (ML) models that directly target ranking problems. Below, after establishing preliminaries and introducing the Similo and VON Similo algorithms, we then present the ML algorithms we consider in the following, e.g. Learning-to-Rank models as well as simple classification algorithms as baselines.

3.1. Preliminaries

Similo is a multi-locator based approach for GUI-testing of web applications. The objective of Similo is to find the candidate web element with the highest similarity to the target one. Similo uses more information than a single locator, similar to the multi-locator approaches discussed in the Related Work section. However, instead of using multiple single-locators, like the solution from Leotta et al. (2016), Similo finds the candidate web element with the highest similarity score by comparing multiple locator parameters from the target with each candidate. Additionally, the approach can be extended by using the concept of visually overlapping nodes (VON) that exploits the hierarchical structure of how modern web applications are constructed in the document object model (DOM) (Wood et al., 1998) to improve performance. DOM nodes that overlap visually (i.e., when comparing their location and size) do, according to the VON concept, represent equivalently valid targets.

For the remainder of the paper, we will consider a scenario in which the Similo algorithm is applied to find web elements in the current version of the SUT (named *v2*) based on the identification of web elements from a previous version of the SUT (named *v1*). We adopt the following nomenclature:

- **Equivalent**: web elements that are considered as equivalent with any other, according to the Visual Overlap heuristics that are defined for the VON Similo algorithm;
- **Target**: web element to be identified, taken from the DOM hierarchy of v1 of the SUT;
- **Target set**: set of the web elements that are equivalent (according to the VON Similo heuristic) to the current target, in the DOM hierarchy of v1 of the SUT;
- **Candidate**: web element, in v2 of the SUT, that is checked with the Similo algorithm to evaluate its as a match to the target web element;
- **Match**: a web element(s) that is/are provided by the Similo algorithms as the corresponding one(s) to the target web element, which also represent the output from the algorithms. If a ranking approach is used, the algorithms return a list of possible matches ranked by similarity rather than a single match. In this case, the candidate with the highest similarity score (the most likely match) is placed at rank 1 of the output list.
- **Oracle (Correct Match)**: candidate web element in v2 that is associated to the target (set) actually equivalent to the target (set) in v1. The oracle was, in this experiment, defined during prior manual exploration of the application and is used to verify the correctness of the matches provided by the algorithm.

3.2. Standard similo approach

This section provides an overview of the standard Similo approach. The interested reader can refer to the original paper discussing the technique for additional details (Nass et al., 2022).

Similo is a multi-locator approach based on a weighted similarity score computed on the difference between the locator parameters of the web elements on the current web page (candidates), and the locator parameters of the sought target element.

To evaluate the correctness of a candidate as a match for the current target web element, Similo uses 14 different locator parameters with corresponding operators and weights, as summarized in Fig. 1. The locator parameters Tag, Class, Name, Id, HRef, Alt, XPath and ID relative XPath, Location, Visible Text are collected directly from the DOM-tree of the web page. *IsButton* is a derived boolean parameter, that is set to true or false according to the values of the attributes Tag, Type, and Class. Neighbor Text contains a space-separated concatenation of words collected from the visible text of nearby web elements. Specific comparison operators that return a value between zero and one (or exactly zero or one) were selected for each locator parameter. Some locator parameters were compared by the Java method equalsIgnoreCase (e.g., Tag, Name, and Id). Others were compared using Levenshtein distance, word comparison, or Euclidean distance. In Similo, the locator parameters were divided into two groups based on a related study by Kirikuni et al. which identified what locator parameters have higher stability and uniqueness. Based on this work, we assigned a weight value of 1.5 to the most stable locator parameters (regular lines in Fig. 1) and a value of 0.5 to the remaining ones (dashed lines in Fig. 1). The weights were selected based on expert judgement for the SIMILO algorithm, without empirical assessment of the effects of their selection on the provided results.

In the original version proposed for Similo, the algorithm selected a single candidate web element from v2 – the one with the highest similarity score – as the *Match* for a given target web element in v1 of the web application. Since the algorithm provided a matching element for every target, it is only possible to have one of two results when evaluating the correctness of the matches: a match is considered a true positive if the candidate web element is the correct selection for the current target (i.e., the oracle), and a false positive otherwise.

In addition to the standard behavior that always returns a single match, the Similo algorithm can be used as a binary classifier by comparing the Similo score with a fixed threshold (which serves as a parameter for the algorithm). If the computed score is higher than the threshold, the candidate web element is considered a match for the target element.

3.3. Visually overlapping nodes (VON) Similo

VON Similo is an extension of the original Similo algorithm, which considers the possible visual overlap between different web elements in a given page hierarchy (Nass et al., 2023a). A characteristic of modern web applications is that they are built as a hierarchical structure that is modeled within the DOM. As an example, it is possible that a div tag contains a button which in turn contains a text label, all these elements sharing the same visual portion of the screen. From a DOM perspective, each element is considered a unique entity, and it is described by a DOM node identifiable through an absolute XPath. However, multiple web elements can be rendered – from a visual perspective – inside a single entity that is visualized on screen (e.g., a button) due to their relative size and placement.

We call the single human-visible rendered entity a *widget*, and we refer to the set of overlapping web elements as *equivalent web elements*.

Fig. 2 visualizes this many-to-one connection, showing how web elements can be contained in other web elements yet, from a visual perspective, occupy the same – or a very similar – area of the screen. We refer to this phenomenon as visually overlapping nodes (VON), where property-based localization approaches, like Similo, can utilize VON to increase the number of correctly located candidate web elements. In the sample case, while web elements W2 and W3 are represented with different XPaths in the DOM, both visually point to the same (or almost equal) screen area or web element. This phenomenon implies that both web elements (W2 and W3) can be equally correct matches for the same target element or, vice-versa, equally correct targets for the same candidate web element. By using this heuristic concept of equivalence, any property-based localization approach can use VON to increase the

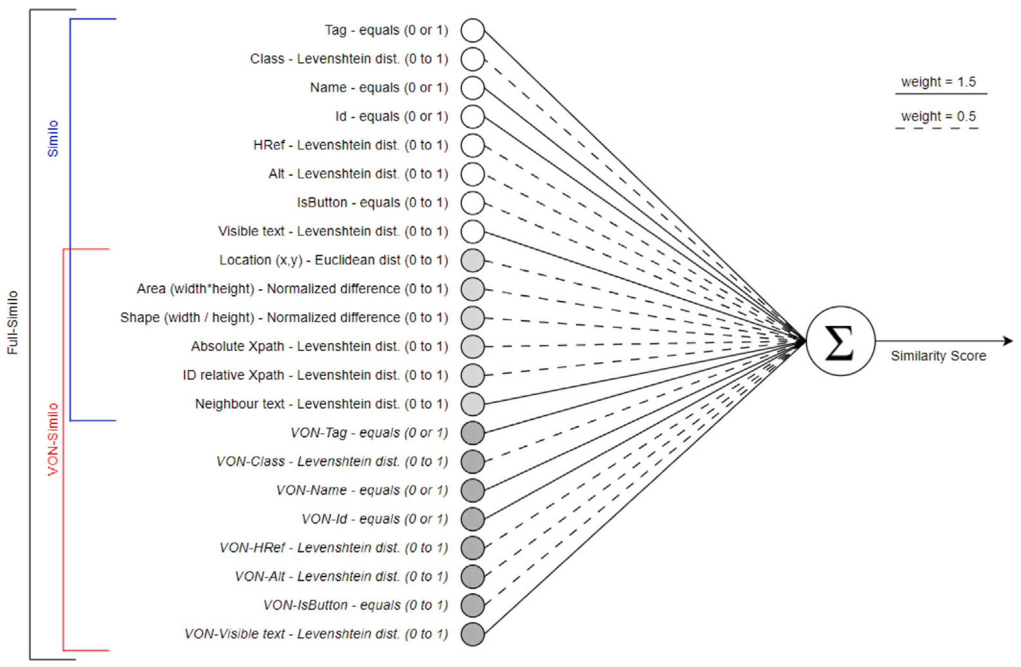
Full-Similo
Similo
VON-Similo
Tag - equals (0 or 1)
Class - Levenshtein dist. (0 to 1)
Name - equals (0 or 1)
Id - equals (0 or 1)
HRef - Levenshtein dist. (0 to 1)
Alt - Levenshtein dist. (0 to 1)
IsButton - equals (0 to 1)
Visible text - Levenshtein dist. (0 to 1)
Location (x,y) - Euclidean dist (0 to 1)
Area (width*height) - Normalized difference (0 to 1)
Shape (width / height) - Normalized difference (0 to 1)
Absolute Xpath - Levenshtein dist. (0 to 1)
ID relative Xpath - Levenshtein dist. (0 to 1)
Neighbour text - Levenshtein dist. (0 to 1)
VON-Tag - equals (0 or 1)
VON-Class - Levenshtein dist. (0 to 1)
VON-Name - equals (0 or 1)
VON-Id - equals (0 or 1)
VON-HRef - Levenshtein dist. (0 to 1)
VON-Alt - Levenshtein dist. (0 to 1)
VON-IsButton - equals (0 to 1)
VON-Visible text - Levenshtein dist. (0 to 1)
weight = 1.5
weight = 0.5
Σ
Similarity Score

Fig. 1. Graphical representation of the computation of similarity score between two different sets of locator parameters. The attributes in blue and red are, respectively, used by the Similo and VON-Similo versions of the algorithm. The Full-Similo attribute set considers the union of the two attribute sets. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

number of DOM nodes that can be matched when identifying the same web element.

Given a web element W1, we define the set of equivalent web elements $e_0,\ldots,e_N$ as the set of web elements that satisfy the following properties:

1. The ratio between the overlapping areas of the web elements on the screen, and the union of the areas of the two web elements, is higher than a set threshold value. This threshold is computed as,
$$\frac{\cap(R_1, R_2)}{\cup(R_1, R_2)}$$
where: R_1 and R_2 are the rectangles occupied on the screen by the two web elements; the set intersection symbol indicates the size (in pixels) of the common area occupied on the screen by R_1 and R_2, and the set union symbol indicates the size (in pixels) of the union of R_1 and R_2.
By experimenting with different threshold values to identify visually overlapping nodes, we finally selected 0.85 as a value for the threshold that allowed us to avoid a definition of visual overlap that was too loose (thereby considering visually separate GUI elements as overlapping) or too strict (thereby finding none or a few visually overlapping nodes).
2. The center of the web element W2 is contained in the rectangle R1.

Hence, rather than relying on finding one specific DOM node, or absolute XPath, any located DOM node that belongs to the same visual element is deemed a correct match. The presence of equivalent web elements, therefore, creates a *target set* for a single target made up of all web elements that are equivalent to the target.

In addition to the extension of the target set, VON Similo also substitutes the attributes compared in the original Similo, by incorporating inside the attribute set of each web element the information of all its equivalent web elements. The following function is used to compute the equivalent-based set of attributes:

This approach mitigates test execution fragility by, as mentioned, increasing the number of candidate DOM nodes that can be correct matches (i.e., corresponding to oracles) when identifying one and the same web element. Essentially, the approach will be more robust as long as only one or a subset of these nodes change between two revisions.

The addition of the VON concept allows the addition of a new set of locator parameters (e.g., VON-tag, VON-id, or VON-xpath) stored and utilized by the Similo algorithm. Each VON locator parameter is no longer a single value but is instead a list of parameters, that are collected from all the visually overlapping web elements of a DOM node. Therefore, in addition to the extended target set, the VON Similo algorithm also allows the extension of the attribute set by considering the attributes with information from equivalent nodes.

To compare the attributes of the extended VON Similo attributes, the following comparison function is used: given the set ep_1 (set of values of the attribute for w1 and all its equivalents web elements: $p_{1.1}, p_{1.2}, \ldots, p_{1.N}$) and ep_2 (set of values of the attribute for w2 and all its equivalent web elements: $p_{2.1}, p_{2.2}, \ldots, p_{2:M}$), the result of the comparison is the highest value among the individual comparisons of all possible combinations of values from ep_1 and ep_2.

Fig. 3 presents a visualization of the process of how visually overlapping nodes are utilized in VON Similo. In the first step, denoted A in the figure, a set of target web elements (denoted $T_x \in TS$) and candidate web elements (denoted $C_y \in CS$) are available, where $|TS| \leq |CS|$. In the second step, denoted B in the figure, a pre-analysis of TS and CS is performed, clustering all target and candidate web elements according to the visual web elements they are associated with, using the formula presented previously in this section. The outcome of the pre-analysis is clusters TS_1-TS_i and CS_1-CS_j containing components with overlapping target areas on the screen but otherwise with an unknown overlap in terms of locator properties. In step 3, denoted C in the figure, each target web element $T_x \in TS_i$ is compared to every other candidate web element $C_y \in CS_j$, and a similarity score is calculated. After the comparison, the maximum similarity score of each cluster is kept and associated with all target web elements T_n-$T_m \in TS_i$, resulting in a mapping between T_i and C_j as visualized in the last step of the figure, denoted D. This mapping implies that (from a DOM perspective) a given target web element T_i may not be mapped to the candidate component $C_i = T_i$ that was initially used when the target was set in the previous

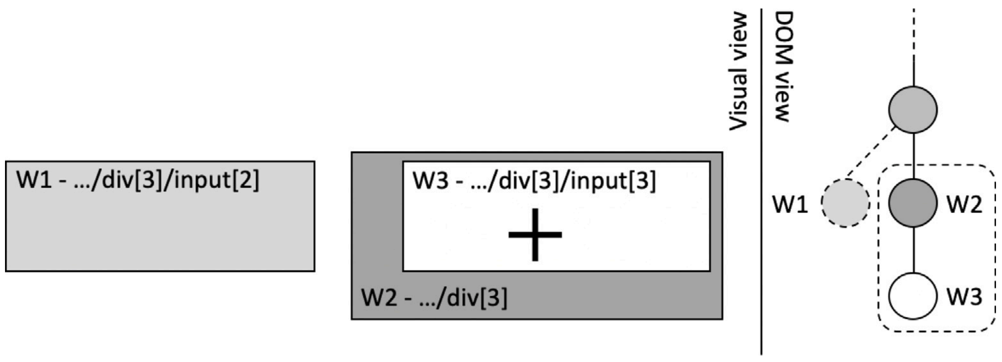

Fig. 2. A visualization of a hierarchy of web elements represented both visually and from a DOM perspective. It shows that although W2 and W3 are unique entities, they appear to be the same visual component or, at least, overlap visually. Therefore, for the VON concept, w2 and w3 are considered as equivalent web elements, and are not equivalent to w1. Note that, for readability of the image, the exceeding contour of w2 is significantly larger than that of w3, while in common layout hierarchies several widgets share the exact area on screen.

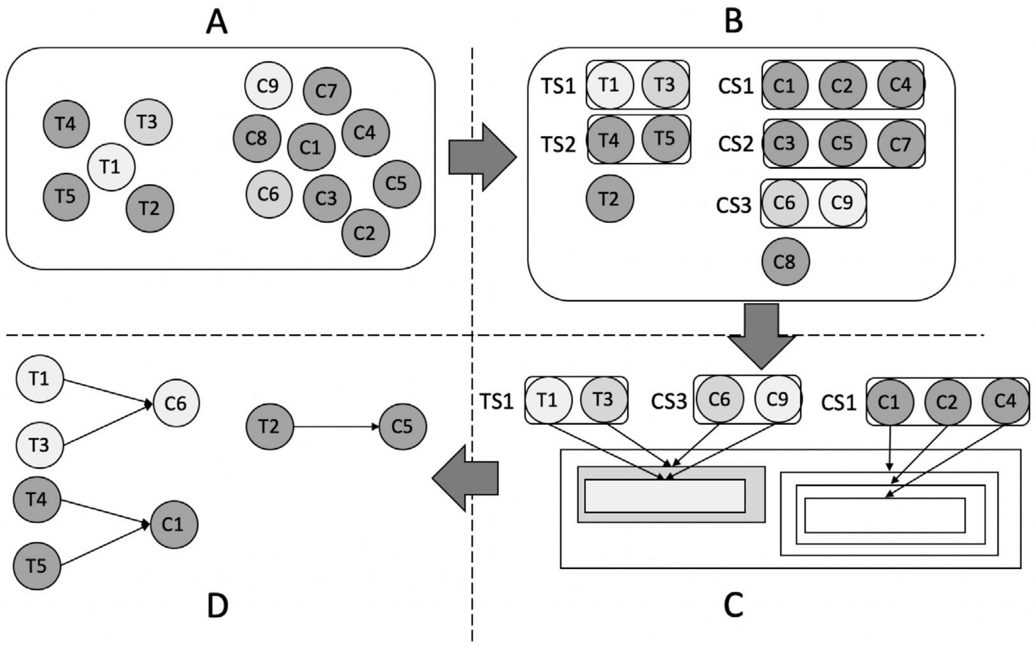

Fig. 3. Visualization of how visually overlapping nodes are implemented in VON Similo.

version of the SUT. In Fig. 3, T_1 is equivalent to C_9 but is mapped to its parent component C_6. However, from a visual perspective, this is irrelevant since both C_6 and C_9 point to the same visual area, T_1. The benefit of this approach is that the number of valid matching candidates increases, implying that for clusters where a target web element cannot be mapped to a suitable candidate using the original Similo approach, a functioning candidate can still be associated.

In Nass et al. (2023a), we evaluated the benefits introduced by using the VON concept, and performed the comparison between the original and VON Similo. In this evaluation, both the algorithms were compared as binary classifiers, evaluating the correctness of the classification of candidate web elements as corresponding to target web elements. In this evaluation, standard accuracy and precision measures for binary classifiers were applied by computing the percentage of true and false positives and true and false negatives in classification.

3.4. Limitations of the existing Similo algorithms

While the evaluation of the previously defined Similo algorithms provided encouraging results compared to the prior state-of-the-art, there are several limitations that might reduce their effectiveness in real testing scenarios.

First, both algorithms employ fixed and manually selected weights to calculate the combined similarity score for all locators. Although these weights are in line with existing literature in the field (Kirinuki et al., 2019) and are compatible with the distribution and variability of web elements in common web applications (Nass et al., 2023a), having a dynamic and data-driven adaptation of the weights to relevant sets of targets and candidates can be a significant improvement for the performance of the algorithms.

Both Similo and VON Similo algorithms provided a similarity score for each candidate web element, given a target, and therefore such similarity scores could be used to provide a list of all the candidates ranked by similarity. The initial implementation of the two algorithms, however, did not provide such a list, only providing a single result, the most similar candidate, for each target web element.

In a second evaluation of the approaches, we evaluated the accuracy of the algorithms when considered as Binary Classifiers. Both algorithms proved efficient in such context, providing high measures for Accuracy, Precision, and Recall. However, using the algorithms as binary classifiers would imply, in an automated setting, potentially returning a list of multiple candidate web elements as matches for the original target (i.e., all candidate matches yielding a score above the defined threshold for a match). In these cases, especially if the algorithms are employed for web element localization in automated testing procedures, it may be beneficial to obtain a ranking of all the candidates identified as matching (by the binary classifier) to allow automatic selection of the *best* candidate for a given target. In a way, the ranking perspective is thus the more natural one; however, in prior work the binary classification and select-most-similar uses of the algorithms did not consider the full list of ranked candidates; a missed opportunity. Having a ranked array of candidate widgets as an output is seen as a better opportunity of robustness for locating algorithms in practice, since they allow for continuing the execution of test sequences in case the top-ranked widget is not the exact oracle and cannot be interacted in the webpage.

The use of a threshold similarity score needed for binary classification poses another limitation. When none of the candidates yields a score above the threshold after comparison while there should be a correct match with a given target. In these cases, the application of the algorithm provides a *false negative* result, since no matches are provided for a valid target. This *no-match* situation is detrimental for both completely automated test procedures, in which the execution of the test case would end, and manually-supervised test procedures, where the tester would benefit of the indication of several possible candidates before deciding for the absence of a match.

Finally, the match threshold is an additional hyper-parameter of the algorithm that would require careful tuning for each individual (type of) SUT where the algorithm is applied.

The mentioned limitations of the previous experimental setups and proposed use scenarios suggest that viewing the problems as a ranking of all of the candidate web elements is the more natural and less limiting one. Tuning the models in a data-driven manner is also natural and should be investigated. In the following, we introduce the machine learning algorithms and models we have considered and experimentally evaluated.

3.5. Machine learning approaches for ranking web elements

There are specific ML algorithms developed for ranking problems, e.g. Learning-to-Rank (Liu et al., 2009). However, even basic ML algorithms used for binary classification actually predicts the probability of a match after which a threshold is applied to give a binary output. This is in line with how we previously proposed to threshold and use the Similo approaches as binary classifiers. We thus have selected both a Learning-to-Rank algorithm and included two basic ML algorithms used for binary classification as baselines. This should allow us to better understand if any benefits from these comes mainly from their machine learning, data-driven nature, or from being specifically developed for ranking problems.

There are many Learning-to-Rank algorithms and models in the ML literature, but much like models based on gradient boosted trees frequently are among the best performing, traditional ML algorithms (Nielsen, 2016), they have also shown good performance for test case prioritization when used as a Learning-to-Rank approach (Lin et al., 2021). We thus use the mature and high-performing XGBoost library (Chen et al., 2019) in its Learning-to-Rank (LtR) setting.

The XGBoost package includes multiple different objective (loss) functions, corresponding to the major LtR approaches in the literature. We thus include three different objective functions, to cover a breadth of different LtR approaches. We also consider if increasing the size of the model itself has any major effect on performance. We thus evaluate a total of six (6) different LtR variants: two different model sizes (the default of 100 sub-trees and one setting with 200 sub-trees) times three different objective functions:

- *NDCG: (Normalized Discounted Cumulative Gain)*: a ranking quality metric that evaluates how closely a given ranking aligns with an ideal ordering, where all relevant items are positioned at the top. NDCG at position K is calculated by dividing the Discounted Cumulative Gain (DCG) by the ideal DCG, which represents a perfect ranking scenario. DCG quantifies the total relevance of items in a list, applying a discount factor that accounts for the decreasing importance of items as they appear lower in the ranking. NDCG is the default objective function for XGBoost;
- *MAP (Mean Average Precision):* a ranking quality metric used for tasks with binary relevance, i.e. when the true score y of a document d to be ranked can be only 0 (non relevant) or 1 (relevant).
- *PW (Pairwise):* it considers pairs of documents and tries to minimize the number of incorrectly ordered pairs. Also known as LambdaMART, because it combines MART (Multiple Additive Regression Trees) and LambdaRank, this is the original version of the pairwise loss function (also known as RankNet).

For the basic ML algorithms used as baselines we have included logistic regression (LaValley, 2008) as well as a random forest variant; both used in the binary classification setting but extracting their predicted probability as a ranking (similarity) score.

We included logistic regression since it is one of the simplest. Its model has the same form as Similo itself, i.e. a simple linear model with a weight per feature. However, while the Similo weights are manually selected logistic regression optimizes the weights based on the training data.

We included the random forest algorithm (Rigatti, 2017) since it is frequently among the most well-performing ML algorithms in evaluations, even within software engineering (Kaur and Kaur, 2018). Specifically, we have used the randomForestSRC R library (Ishwaran et al., 2022), in its default settings, in our experimental evaluation below.

4. Evaluation

The goal of this work is to evaluate the benefits introduced by the application of machine learning and learning to rank algorithms in the context of similarity-based multi-locators for web element localization and web application GUI testing.

In our evaluation, we always consider the extended *target set* obtained by applying the VON concept.

The evaluation that we performed is manyfold. We perform an evaluation of the impact of weight variation on the algorithm, to understand the benefits that could be introduced by the application of machine learning-based algorithms to automatically infer the weights from web element distributions.

Building on top of the first experimental results of the evaluation of Similo and VON Similo as a binary classifier, we evaluate the benefits provided by the extended attribute sets. Finally, to address the limitations discussed in the previous section of the paper, we evaluate the applicability of learning to rank algorithms for web element localization of Similo.

In particular, we consider three different approaches with static weights:

- *Similo*: the original approach using the standard attribute set (14 attributes considering no equivalent attributes);
- *VON Similo*: the VON approach using the enhanced attribute set (14 attributes, VON attributes in blue in Fig. 1 used instead of correspondent non-equivalent ones);
- *Full-Similo*: both original and enhanced VON attributes are used (22 attributes, all in Fig. 1).

Since our previous works demonstrated the benefits of extending the target space, we will perform all our evaluations with the VON target space (i.e., considering equivalent web elements for both targets and candidate web elements).

4.1. Research questions

To address the goals of our evaluation, we formulated the following set of Research Questions:

- **RQ1:** What is the robustness of the web element localization strategies based on the Similo, VON Similo and Full-Similo attribute sets?

The aim of RQ1 is to evaluate the robustness of three different combinations of web element attributes, without applying machine learning algorithms, to infer the weight for each attribute.

The original SIMILO and Von-SIMILO algorithms utilized two fixed weights (0.5 and 1.5) for the different attributes used in web element comparison. Although these weights guaranteed satisfying results, it was worth performing an additional investigation to understand whether the algorithms were sensitive to modifications in the weights applied to the attribute sets.

For that purpose, we sample random weight sets and run the algorithms on a common set of targets and candidates to evaluate the impact of the weights on web element localization.

- **RQ2:** What is the benefit of applying traditional Machine Learning or Learning to Rank algorithms to optimize web element localization?

The aim of RQ2 is to verify whether Machine Learning and Learning to Rank algorithms can improve effectiveness when applied to the weights for web element localization over a set of manually selected oracles for pairs of different versions of the selected benchmark SUTs. To this purpose, we compare multiple ML and LtR approaches for weight optimization, and we adopt standard metrics for the evaluation of the ranked results.

4.2. Methodology

This section illustrates the methodology that was used to collect the experimental subjects, details the experimental evaluation and lists the measurements taken to answer each research question in our study design.

A preliminary step for our methodology was the construction of a data set of pairing matches between target and candidate web elements taken from two different versions of each application in our sample set. These pairings were used as an oracle to perform the evaluation which was performed according to the following steps:

1. *Application Collection*: The sample of subject applications used in our evaluation was reused from the study by Nass et al. (2023a). The set consists of the 40 top-rated web applications in the United States from the Alexa ranking web application.[1][2] We only excluded mirrored web applications to improve the diversity of the sample and applications with adult content due to ethical reasons. The details of all the web elements within the selected applications can be found listed in the replication package (Coppola et al., 2024).
2. *Application Version Selection*: to acquire two versions for each SUT (denoted *v1* and *v2*) with enough differences to be suitable for our study, we used the Internet Archive web application.[3] We replicated a design employed by Leotta et al. where the newer version (*v2*) is represented by the currently available application online at the time the data set was created (in this case, published in December 2020). An R months older version of the application (*v1*) was also acquired from the Internet Archive, where R randomly varied between 12 and 60 months, 36 months on average backward in time. This difference in time between the application samples was perceived as adequate to see graphical and functional differences, enabling evaluation of the web element finding robustness of the approach for both minor and significant changes. The assertion that 36 months, on average, was enough was verified through manual inspection. Note that minor application changes are perceived to have higher construct validity for regular operations of the approach in practice, e.g. in a continuous integration environment. Still, more significant application changes cannot be excluded from the dataset since such occur when companies re-brand or make more extensive technological updates to their web applications.
3. *Correspondent web element Selection*: we manually selected widgets from both versions of each application's homepage, e.g., its start page. This design choice may have delimited the generalizability of the results, but analysis of the collected sample suggests any such effects to be minor. Furthermore, this sampling became necessary since the Internet archive only stores static pages, meaning that javascripts, databases, etc., do not pertain to full functionality. Specifically, widgets were sampled from each SUT according to the following criteria: (i) it is possible to perform actions on the widget (e.g. click or type); (ii) it is possible to use the widget for assertions or synchronization; (iii) the widget is associated with a core feature(s) of the SUT; (iv) the widget was present in both v1 and v2 of the SUT's homepage. This selection was performed manually, resulting in a set of 442, 1-to-1, matched pairs of corresponding widgets that were applicable to the experiment.
4. *Equivalent web elements Selection*: The 442 pairs was then automatically extended by applying the web element equivalence definition of the VON concept. This effectively expanded the 1-to-1 matching into a 1-to-many matching, extending the oracle from 442 matching pairs to 1163 matching pairs.

4.3. Analysis method

To evaluate the robustness (our effectiveness metric) of Similo, VON Similo and Similo-ML, the three algorithms were executed on the equivalent pairs of web elements between consecutive releases of the SUTs, to identify whether the original target web element would be correctly associated to a corresponding candidate web element.

The evaluation of all algorithms was performed as a ranking problem where all web elements were ranked for similarity to a target web element. To achieve this, minor changes were made to Similo and VON Similo which changed their output from returning a single, best matching, web element to a list of the most similar web elements.

Below, we list the metrics that were used to evaluate the results. All the metrics refer to an output in the form of an ordered list of possible matches provided by one of the variants of Similo, following the assumption that a lower rank (closer to 1), is a better match for the target web element.

- *MeanRank*: the mean rank is computed as the average of the ranks of the executions of a given ranking algorithm over the whole set of targets. The *rank* is defined as the position, in the ranked list of candidates, of the first candidate matching that matches the target (note that in the case of the Similo algorithm there is only one matching candidate). The mean rank is therefore computed as the average of the ranks (average position of the candidate web elements) obtained for all the searched targets. Note that due to the VON concept, the list of ranked candidates can contain more than one possible match. This metric is considered the best predictor for our purposes since we seek their capability in avoiding the worst case (where no match is found) over optimizing that the correct web element is always ranked as the first item in the output list;

[1] http://www.alexa.com.

[2] Since the experiment, the website providing the site ranking (i.e., alexa.com) has been discontinued, and is no longer publicly available.

[3] http://web.archive.org.

- *MaxRank*: the MaxRank is the highest rank (worst performance) obtained for one of the targets in the target set. It can be considered the worst case scenario for a given ranking algorithm;
- *PctAtN*; percentage of cases in which the first correct match is ranked among the first N items in the output list provided by the algorithm. Note that PctAt1 corresponds to the recall of a binary classifier where we have a true positive if, and only if, the first item in the output list is a correct match.

Since the purpose of the experiments are to evaluate the ranking behavior of the algorithms, we do not present the results of their behavior as binary classifiers in this manuscript. Instead we refer readers interested in such results to our previous work (Nass et al., 2023a).

4.3.1. Robustness evaluation

To answer RQ1 we want to understand how sensitive the different Similo algorithms are to the choice of weights associated with their attribute sets, as presented in Fig. 1. The weights that were used in the original variants of both Similo and VON Similo were either the distinct values 0.5 or 1.5, depending on locator.

In this experiment, we sample the weights to optimize the output performance, allowing these values to vary uniformly from 0.0 to 1.5. We allow values of 0.0 since that would correspond to a locator not providing value to the localization of a web element. The top value of the range is indeed arbitrary, and was kept at 1.5 for continuity with the previous work.

Weights were then sampled randomly 1000 times for each of the three algorithms, original Similo, VON Similo, and Similo Full, respectively. Together with the original weights used by the three algorithms, we acquired 3002 unique weight vectors which we then evaluated using 10-fold cross-validation.

For each of the sampled weight vectors per locator set we then find the best and the worst in terms of output performance (in relation to the metrics discussed above) and present the results. To provide a summary of the results, we also calculate the mean values per output metric.

4.3.2. Evaluation of ML and LtR algorithms

To answer RQ2 we want to evaluate the benefits of the application of different Machine Learning and Learning to Rank algorithms for the identification of web elements based on vectors of comparisons of the presented locator parameter sets. As traditional Machine Learning algorithms, we evaluate *Logistic Regression* (from now on referred to as *LogReg*), and *Random Forest*. As Learning to Rank approaches, we evaluate several configurations of the *XGBoost LtR* approach (namely: *pairwise*, *map*, *ndcg*, and *XGBoost-200*).

The results provided by the different algorithms are then compared to the standard, fixed weight approaches. We separate the results based on the attribute sets that were selected among the three alternatives represented in Fig. 1.

To evaluate the ML approaches, we apply the 10-fold validation technique. This involves splitting the available dataset into ten subsets. The split for the 10-fold validation was made by considering all the web elements of individual applications so that the folds were roughly equal-sized. We applied random weights to all folds, and then averaged the results over all the folds. While this is not theoretically necessary when no training is involved, we deemed that valuable to ensure the comparability of the results that are provided for RQ1 and RQ2.

5. Results

5.1. RQ1: Robustness

In Table 1, we report all the evaluation metrics for the original, VON, and Full versions of Similo with one hundred sampled sets of weights each. The complete set of results, including all the different results obtained for each randomly-sampled set of weights, is included in the online appendix of the paper (Coppola et al., 2024).

Of the hundred sampled sets of weights, we report the results for the weight sets optimizing the mean rank and the PctAt1, the average of the metrics over all the random sets, and the worst value for each metric. We also compare these results with the *standard* set of default weights described in previous sections.

In Fig. 4, we report the density of the distribution of the MeanRank obtained by the 3000 random weights combinations used (1000 per version, plus the 3 standard algorithms). In the graph, we overlay vertical lines representing the standard versions of the Similo (red) and Similo-VON (blue) algorithms. We do the same in Fig. 5 for the PctAt1 metric. For sake of brevity, we do not report similar graphs for the other PctAtN metrics, since they had a similar shape to the one for PctAt1.

For the original version of the algorithm, we obtain a minimum mean rank for the best case of 1.77, meaning that in the typical execution of the algorithm, a correct match is found for a target web element at most at the second position of the produced output list of ranked candidates.

With the best weight set to maximize PctAt1, a correct match is provided as the first item of the output list in 90.50% of cases, and in the first 10 results of the output list in 96.60% of the cases.

For VON Similo, we obtain a best mean rank of 1.89, and a best PctAt1 of 87.56%. In addition, we get a PctAt3 of 95.25%, PctAt5 of 96.83%, and PctAt10 of 97.96%. As such, we conclude that original Similo outperforms VON Similo in Mean Rank and best achievable PctAt1, while Similo-VON achieves a lower Max Rank (31 compared to 68) and best PctAt3, PctAt5 and PctAt10.

For both Similo and VON Similo, the standard version of the algorithms, with statically set weights of 0.5 and 1.5, generally performed worse than with the best optimized weights, but better than the average random weights.

The implications of these results are manyfold: first, we conclude that the selection of weights has a relevant impact on the algorithms' performance, suggesting that the adoption of learning techniques to infer the best possible set of weights provides a benefit. Second, we deduce that our intuitive, literature-based selection of weights was not well-suited to minimize the MeanRank of the algorithms' output but was instead more suited to placing the correct matching web element as the first item of the output list, i.e., optimization of PctAt1.

In our results, we see that the Full version of the algorithm, which performs 22 different attribute comparisons, had an average performance worse than both Similo and VON Similo, in terms of MeanRank. The best possible MeanRank obtained with the Full version (2.71) was almost one index higher in the output list than the best MeanRank for the other two algorithms. On the other hand, the Full version of the algorithm obtained the best PctAt1 of the three algorithms. The explanation for this result is assumed to be that using 22 attribute comparisons in Similo Full, instead of 14 in the other algorithms, significantly expands the vector space of possible weight vectors. Therefore, we cannot rule out that extending the number of randomly-generated weight vectors would result in a vector (global optima) that could outperform the other algorithms also in terms of MeanRank.

5.2. RQ2: Evaluation of ML and LtR algorithms

Based on the results collected to answer RQ1, we deemed that no significant advantage is provided by considering the full set of attributes for web element location (i.e., using the Full-Similo attribute set). Therefore, we evaluated the application of ML and LtR approaches on the smaller Similo and VON Similo attribute sets.

In Table 2 we report the results of the 10-fold cross validation of the LtR algorithms and traditional ML solutions, applied with, and without, the VON concept. The complete set of results, providing the ranking for all the target web elements and for all the considered methods, is included in the online appendix of the paper (Coppola et al., 2024).

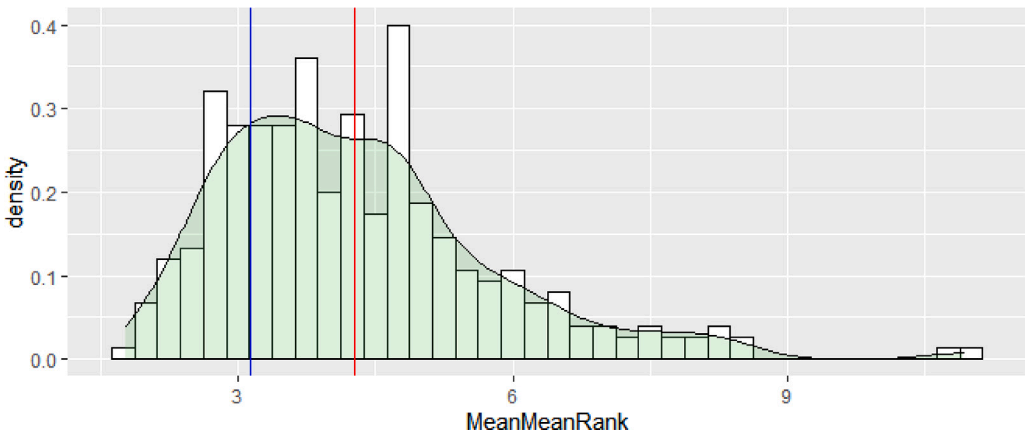

Fig. 4. Distribution densities of Mean Ranks for the 3000 considered variants of the Similo algorithm. Vertical intercepts for Similo - original (red) and Similo - VON (blue). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

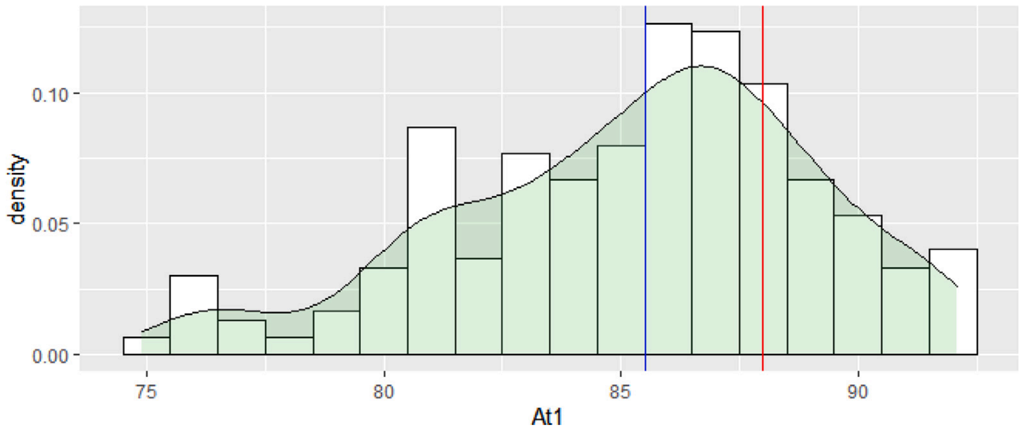

Fig. 5. Distribution densities of PctAt1 for the 3000 considered variants of the Similo algorithm. Vertical intercepts for Similo - original (red) and Similo - VON (blue). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

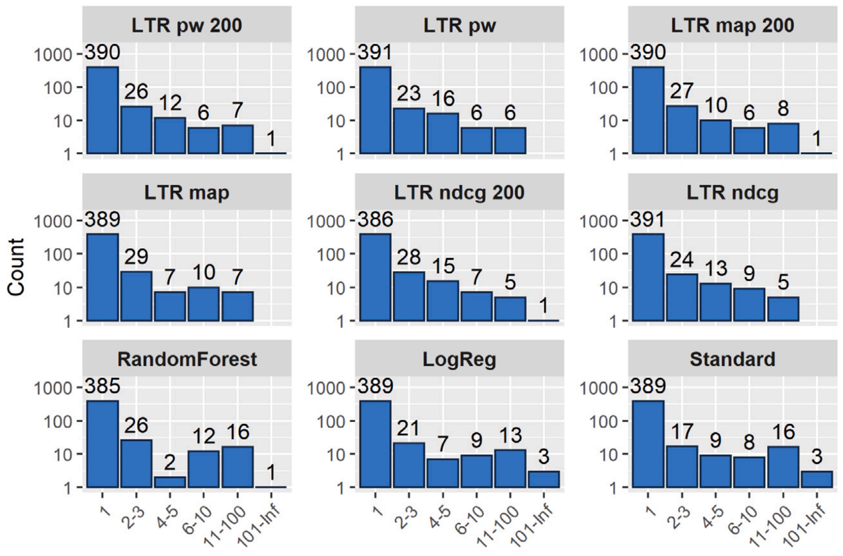

Fig. 6. Frequencies of ranks for the different algorithms with the Similo attribute set.

Table 1
Results of the three algorithms with randomly-sampled weights.

Feature set	MeanR	MaxR	PctAt1	PctAt3	PctAt5	PctAt10
Original						
Best MeanRank	**1.77**	68	81.00	92.08	96.61	98.42
Best PcAt1	3.65	471	90.50	93.44	94.57	96.60
Original SIMILO (Nass et al., 2023b)	4.28	321	88.00	91.86	93.89	95.70
Average all	3.84	322	84.07	90.99	93.38	95.79
Worst all	7.92	777	74.89	85.29	87.33	90.95
VON						
Best MeanRank	1.89	**31**	83.48	95.48	97.29	97.96
Best PcAt1	2.61	286	87.56	95.25	96.83	97.96
VON-Similo (Nass et al., 2023a)	3.13	402	85.52	93.34	95.70	97.06
Average all	3.90	297	83.19	91.74	94.02	96.67
Worst all	8.23	459	75.11	83.48	87.33	91.86
Full						
Best MeanRank	2.71	287	87.56	95.25	96.83	98.42
Best PcAt1	3.84	389	**92.08**	**95.93**	**97.29**	**98.87**
Average all	5.12	373	88.59	94.39	95.81	97.30
Worst all	10.92	485	82.81	89.82	92.31	94.57

Table 2
Results for the evaluation of ML and LtR algorithms.

Attr. set	Method	MeanRank	At1	At3	At5	At10	MaxRank	*Rank*
VON	XGBoost-200 LtR pw	1.937	84.163	**94.796**	95.701	97.511	115	11
VON	XGBoost LtR pw	1.980	86.652	94.118	95.249	97.738	124	13
VON	XGBoost-200 LtR map	1.803	85.52	94.344	95.928	97.738	83	9
VON	XGBoost LtR map	1.785	84.842	94.344	95.701	97.964	78	7
VON	XGBoost-200 LtR ndcg	1.765	85.747	94.344	95.475	97.738	54	8
VON	XGBoost LtR ndcg	1.817	83.71	93.891	95.701	97.511	58	10
VON	RandomForest	1.937	82.579	94.570	95.701	97.059	60	12
VON	LogReg	2.387	85.747	94.344	95.249	96.380	172	14
VON	Standard	2.801	85.520	94.344	95.701	97.059	402	16
Similo	XGBoost-200 LtR pw	1.846	88.235	94.118	96.833	98.190	133	4
Similo	XGBoost LtR pw	1.613	**88.462**	93.665	**97.285**	98.643	53	3
Similo	XGBoost-200 LtR map	1.896	88.235	93.344	96.606	97.964	137	6
Similo	XGBoost LtR map	1.588	88.009	94.570	96.154	98.416	**49**	2
Similo	XGBoost-200 LtR ndcg	1.857	87.330	93.665	97.059	98.643	146	5
Similo	XGBoost LtR ndcg	**1.570**	**88.462**	93.891	96.833	**98.869**	**49**	1
Similo	RandomForest	2.740	87.104	92.986	93.439	96.154	152	15
Similo	LogReg	4.029	88.009	92.760	94.344	96.380	268	17
Similo	Standard	4.278	88.009	91.855	93.891	95.701	321	18

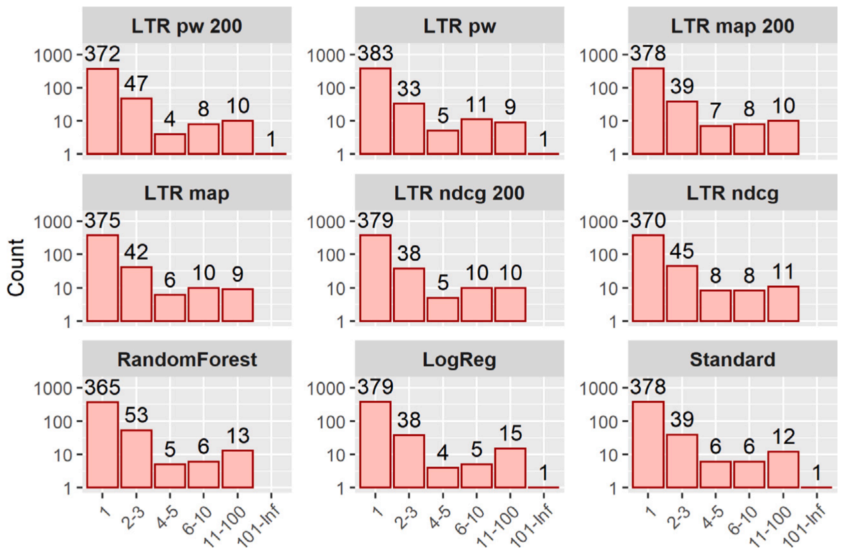

Fig. 7. Frequencies of ranks for the different algorithms with the VON attribute set.

In the table, we report the mean over the folds of the mean rank for each target, the At1, At3, At5 and At10 percentages, and the max rank for each target. The last column of the table indicates the relative positioning of the algorithms in terms of obtained mean rank. In Fig. 6 we report the distribution of the ranks for the 9 algorithms with the Similo attributes. The ranks are distributed in the following bins: 1

(perfect match at first position), 2–3 (contributing to the At3 metric), 4–5 (contributing to the At5 metric), 6–10 (contributing to the At10 metric), 11–100, and 101–442. In Fig. 7 we do the same for the 9 algorithms with the Von attribute set.

The first conclusion that can be drawn from the table is that the VON Similo algorithm with standard weights outperforms the Similo algorithm with standard weights when used to solve a ranking problem. The former, in fact, obtains an average MeanRank of 2.801 (meaning that, on average, the first correct candidate for the target is found at index 3 in the output list of ranked candidates) against a MeanRank of 4.278 for the latter. VON with standard weights also presents a higher At3, At5 and At10 percentage compared to Similo with standard weights. On the other hand Similo, with standard weights, has a better At1 percentage than VON, and a lower MaxRank (321 compared to 402). This last result is principally related to the different nature of the Similo and VON algorithms: by creating a set of equivalents for a given target, the algorithm increases the chances that at least one of the different correct matches is found among the first few positions in the ranked list of candidates. The Similo algorithm instead – since it involves the comparisons only of individual web elements and not web element sets – is more likely to find the web elements at first shot (i.e., higher PcAt1) or fail at placing the correct matches at a high position in the list (i.e., lower PcAtN, with N higher than 1).

With the use of traditional Machine Learning approaches, the VON algorithm still outperforms the original Similo algorithm. We performed the 10-fold analysis with the weights obtained by the application of RandomForest and Logistic Regression. The best result that we get with traditional Machine Learning algorithms is a mean rank of 1.937 for VON Similo, compared to a mean rank of 2.740 for Similo. It is worth noting that Similo achieves a better At1 percentage (88.46% compared to 86.65%) while At3, At5 and At10 are comparable between the two versions of the algorithm. We observe a similar relationship between VON Similo with LogReg-computed weights (MeanRank of 2.387, At1 of 85.747) compared to Similo with LogReg-computed weights (MeanRank of 4.029, At1 of 88.009).

Among the Learning-to-Rank algorithms tested, the best option proved to be XGBoost LtR ndcg for Similo and XGBoost LtR ndcg 200 for Von.. The original Similo algorithm outperforms the VON Similo algorithm, with a mean rank of 1.570 and At1 of 88.462% compared to Von's 1.765, and At1 of 85.747%. The best At3, At5 and At10 percentages are obtained, respectively, for VON XGBoost-200 Pairwise (94.796%), Similo XGBoost LtR Pairwise (97.285%) and Similo XGBoost LtR ndcg (98.869%). The results suggests that the extension of the number of matches and the use of the VON Similo parameters, which have overlap of attributes with the same purpose, it is not beneficial for optimization of the mean rank.

6. Discussion

In this work we have compared two conventional weighted, multi-locator, algorithms for web element identification, Similo and VON Similo, with several learning-to-rank (LTR) algorithms to achieve the same goal.

The results of the evaluation show that there are benefits associated with the choice of algorithm in terms of effectiveness and optimization goal, i.e. mean-rank or rank-at-X.

In the scope of this work, we applied algorithms to optimize the mean ranking of the first correct occurrence of a correct match in the ranked list of selected candidates. These results allowed us to reach a best MeanRank of 1.57 by using the XGBoost LTR ndcg technique, with the original set of attributes of the Similo algorithm. This result is an additional improvement to the best result obtained by sampling random weights for the attributes used by the algorithm, confirming a significant impact of the weight selection on the results of the study. In our previous work (Nass et al., 2023a) we performed – on the same benchmark – an evaluation of the Similo and VON Similo algorithms as binary classifiers. The results discussed in the present paper answer a different ranking problem, and therefore, they are not directly comparable with those previously discussed. In previous results we obtained 95% top accuracy for VON Similo, a figure that, however, leaves the possibility that multiple different positives are found for the same target web element, without any ranking. PcAt1, evaluated for the ranking algorithms, can be considered as mostly equivalent to the recall in a classification problem (since it defines the possibility that a correct match is placed first in the ranking). We see (Table 1) that we can obtain a top PcAt1 of 92% with randomly sampled weights, and >88% with LTR algorithms, while still optimizing for lower MeanRank. These results show therefore that the application of algorithms specifically designed to optimize MeanRank have a limited detrimental effect on one-shot finding of the correct target.

Using LTR, as applied in the experiment, represents a paradigm shift in the approach to web element identification since the algorithms output a list of web elements of decreasing probability of being the sought target. This differs from the traditional approach to web element identification, where algorithms provide binary outputs, i.e. either a found target or not. Hence, if a false-positive is reported—A test case fails to identify an existing target—with the conventional approach, the test case would require manual maintenance. In contrast, LTR provides a list of probable web elements that can be iterated upon in descending order. As a consequence, test cases driven by this approach can perform better than conventional algorithms, mitigating maintenance costs of broken locators, but at the expense of higher test execution cost—Test cases may have to iterate over several web elements of the output to complete execution. In a scenario where iteration is required, in the best case, the target web element is found at the second index of the output list, but, in the worst case, the target, assuming an output list of fixed length, is not an element of the ranked list at all. By comparing the best results obtained with machine learning approaches for ranking (Table 2) and results obtained with random sampling of the weights, we were able to provide an improvement of the MeanRank of 0.20 (from 1.77 to 1.57 for the standard Similo approach). Therefore, it might be possible that a specific selection of fixed weights can provide an efficient-enough compromise for the ranking scenario without requiring the application of ML algorithms.

For longer test cases, i.e. with many steps, which may also have several components with targets not ranked in top positions, this approach will add a significant cost to the test execution time. However, these additional costs should then be weighed against the cost of manual root-cause analysis, maintenance and re-execution of the test case with a conventional web element identification approach. The cost of re-running test cases (by trying to use other web elements in the ranked candidate list) can indeed be exponential if multiple correct matches are ranked high in separate steps of the test cases. While the high percentage of PcAt1 suggests that this might be a remote scenario, a possible solution in real testing environments would be resorting to manual analysis only at the end of a fixed amount of iterations of the algorithm still resulting in failing test cases.

LTR-based web element localization is also associated with algorithm training costs. The algorithms used in this work were based on training data from a general set of websites with attributes common to most websites on the web. However, there is potential to optimize this training for a certain website, or development context, to potentially make it achieve even better results through dynamic weight optimization. Worth noting is that both Similo and VON Similo support weight optimization as well, but currently do not contain any mechanisms to do so automatically, i.e. requiring manual optimization. However, our analysis of the weights' impact on the conventional algorithms' performance, through using random weighting, show that the used weight-vector has an impact on performance but that the algorithms perform well also with manually applied, intuitively set, weights. This result is interesting, as it shows that a targeted, conventional, algorithm can outperform learning approaches for certain software problems.

Furthermore, based on the experimental results, we can derive a clear trade-off between the conventional and learning approaches. In scenarios where cost (e.g., execution and implementation costs) are in focus, the conventional algorithms are more suitable, particularly when also factoring in that the learning approach is associated with additional costs to train the algorithms. However, in scenarios where test case robustness is more important than cost, LTR based web element localization can perform better.

In this study we have focused on evaluating some of the standard and well-performing ML-based LtR approaches. Future work should investigate if there are any benefits to using other search and optimization algorithms and approaches for web element ranking. Inspiration can be found, for example, within defect prediction where Yu et al. (2023) found that unsupervised LtR approaches were among the best ones for ranking software modules.

Another important result from this work relates to VON Similo's use of equivalent web elements, meaning the use of visual overlap of web elements, to expand the target space. Experimental results show that when used with fixed weights, the VON Similo algorithm outperforms the algorithm without the VON concept. In contrast, by optimizing the weights with the objective of minimizing the mean rank, the original Similo algorithm – which does not consider the overlap between web elements – outperforms VON Similo, as shown in the experiment for RQ1.

Finally, we must consider the implications of these results in practice. Robust web element identification is perceived as one of the fundamental challenges of automated GUI-based testing, and has been so since the 1980s (Nass et al., 2021). The results of this work are promising, showing that the considered algorithms may provide a maximum 92.08% of correct match at first position (Full similo with random weights) and 98.8$ of correct matches within the first 10 positions in the ranked list of candidates. In the context of the experimental subjects that were used, this translate to 406 to 436 out of 442 target web elements matched with the manually-identified oracles. Although impressive, consider that even at 98 percent success-rate, we can expect two failures every 100 automated GUI interactions. Since GUI interactions can have a one-to-many mapping with GUI test cases, a failing interaction may have an impact on multiple test cases — as an extreme example, if a failing interaction occurs in the login or main screen of an application, all the test cases involving a login will fail. In a test suite in the order of hundreds of test cases, the cost of root-cause analysis and maintenance may therefore still be significant.

Another results that emerged by analyzing all the combinations of weights that were used, is the lack of combinations that are able to provide 100% PctAt1, i.e. perfect matching for all the targets in the considered web applications. We can motivate this result with two (not mutually exclusive) explanations: (i) the properties of some target web elements underwent significant changes between the two considered versions of the application, and could therefore not be identified based only on layout properties without considering their visual appearance; (ii) since the attribute weights were optimized for an entire run, over all targets in our dataset, it is possible that they could not provide globally optimal behavior for all individual, specific cases.

7. Threats to validity

Internal validity threats concern factors that may affect a dependent variable and were not considered in the study. The main internal validity threat in our study is related to the fact that we have considered couples of releases where the time span between v1 and v2 is variable. This randomization of the time span between releases allows us to keep into account more variability for the web elements composing the SUTs. On the other hand, it is possible that the time component has a significant effect on the variability of the attribute values that we did not take into account. Longitudinal studies on the variability of the attributes during the evolution of a web application would be needed to clarify this issue.

Another internal validity threat is a consequence of how the VON concept is utilized for the creation of the candidate set. Since the VON concept allows the generation of a set of multiple candidates, all of which will be considered as valid matches if the threshold approach is used, it is possible that using the algorithms without ranking would result true positives that are completely unrelated to the actual target web element.

The concept of equivalent web elements in VON-Similo is based on a concept of closeness to a given web element on the screen where the test is captured or executed. The distances for the selection of equivalents are relative, so there is no impact of page and web element size on the selection of web elements. However, if changes in the viewport size result in a modification of the arrangement of web elements on the screen (especially if some web element is completely removed), the set of equivalents for a given web element will vary. As a consequence, the results of the application of the algorithm may vary with possible changes in the viewport where the test cases are defined (i.e., on v1 of the web application).

Several threats to validity are related to the design of the algorithm and the selection of ML/LTR techniques and parameters. Albeit the selection that we performed for LTR/ML algorithms is grounded in the literature (Yu et al., 2023), there are additional feature-selection approaches (e.g., PCA) that could provide beneficial results if applied in a context similar to the one where SIMILO is used. Without further comparisons, we cannot conclude that the best algorithms among those that we selected are the optimal LTR algorithms for ranking of pairs of corresponding candidate and target web elements. For RQ1, we relied on the selection of random sets of weights for the attributes used in the SIMILO algorithm, with the objective of assessing whether it could be possible to obtain relevant variations in the outcome of the algorithm by varying the standard weights. The results that are reported for RQ1 in Table 1 suggest an important impact of the weights on the ranking results. Although the number of random selections used for the selection is high, it is however possible that combinations able to achieve significantly better results exist even if only random weight selection is applied.

External validity threats concern the generalizability of the results. Although we selected a relatively large number of experimental subjects, the selection was made based upon the popularity of the web applications according to use. It is possible that some categories of web applications (or web elements composing them) are underrepresented in our sample and therefore our results may not be generalizable to them.

Conclusion Validity concern incorrect conclusion from the observations performed in the study. A possible threat to the conclusion validity of this study is related to the fact that the algorithm was not evaluated in a live setting, i.e., in the execution of real test cases of web applications. However, the purpose of all the versions of the SIMILO algorithm have a final purpose of pairing a candidate web element against a set of targets, to find the correct web element to interact with during the execution of a test case. The atomic operation of ranking and finding the best target web elements for any candidate is not influenced by the execution context, since the attributes that are compared for the web elements can be extracted at any given moment during the execution of a test sequence, even if they are changed by modifications in the page structure and content. Moreover, the web element finding operations in a typical test case for a web application are independent from each other. Therefore, even though entire test sequences would have a lower frequency of correct execution than the accuracy of correctly ranking individual web elements – as motivated and discussed in the discussion section – we do not deem the offline execution of the experiment as an invalidating threat for our study. The integration of the equivalent concept introduced by VON in layout-based testing tools will also need additional development effort, to define mechanisms to correctly execute mouse and/or keyboard operations over a set of equivalent web elements in the current layout hierarchy. The precision of these mechanisms in executing the correct operations on test scripts might require additional validation in future research efforts.

8. Conclusions and future work

In this paper we analyzed the benefits of the application of Learning to Rank algorithms to similarity-based locators for web application testing. To this end, we have set up an experiment with Similo, a solution that was previously evaluated with positive results as a binary classifier (i.e., with the objective of defining if a pair of web elements in separate releases of a web application are a match).

The results of the utilization of Similo in a ranking context are promising and show that Learning to Rank approaches can be applied with positive results to find matching web elements in web application testing, with very high percentages of web elements ranked at first position (maximum of 88.46%) or among the first three (maximum of 94.80%). These results suggest that a similarity-based approach can be highly dependable in a context where a low percentage of matching errors are still accepted (e.g., in a supervised context, or for test sequences with few interactions with the SUT). Overall, the results of the present study held promise for the application of ranking approaches in the context of property-based web application testing. To the best of our knowledge, no other approaches in the literature has been evaluated for its ability to provide a list ordered by similarity for the selection of web elements to execute test case steps. The Similo and VON Similo approaches can still be compared to other algorithms existing in the literature, e.g., Robula+ (Leotta et al., 2016) or Montoto (Montoto et al., 2011). Such comparison holds especially true if only one-shot locator identification is considered, i.e. the ranked list of candidates is ignored in favor of only using the top element of the returned list of candidates. Our previous research efforts (Nass et al., 2023a) proved empirically that the base Similo algorithm has a higher precision in one-shot locator identification. It is worth underlining that any available algorithm that involves the computation of a score for candidate widgets can be modified, in order to provide a list of ranked possible matches as output instead of a single match. Such modification would allow the results of other algorithms to be compared to the LtR Similo approach described in this paper.

As future work, we envision the use of additional machine learning techniques (e.g., Reinforcement Learning), and include additional attributes and characteristics of the web element, for instance the visual appearance of the GUI elements that is considered by now only for what respects the size and position. To evaluate the applicability of the approach without preliminary learning operations, we aim at defining globally-learned parameter values and test their capability in providing precise web element evaluations over diverse sets of web applications. The inclusion of additional attributes can also be paired with the adoption of techniques to perform dimensionality reduction (Palo et al., 2021), such as PCA (Principal Component Analysis), or an ML algorithm that does feature selection as part of its training, such as LASSO (Least Absolute Shrinkage and Selection Operator), to improve the efficiency of the algorithms and reduce the presence of redundancy among features. We also plan to provide a model for the costs of a wrong match in web element location tasks, and use such instrument to evaluate our LtR and ML approaches. Finally, we envision the integration of the SIMILO approach in existing layout-based GUI testing tools to validate the benefits that can be obtained in terms of robustness of the generated test cases and reduction of test case executions failing due to changed locators.

CRediT authorship contribution statement

Riccardo Coppola: Writing – review & editing, Writing – original draft, Methodology, Investigation, Formal analysis, Data curation. **Robert Feldt:** Writing – review & editing, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Michel Nass:** Writing – review & editing, Conceptualization. **Emil Alégroth:** Writing – review & editing, Writing – original draft, Validation, Supervision, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Emil Alegroth, Michel Nass reports financial support was provided by KKS foundation. Robert Feldt reports financial support was provided by Swedish Scientific Council. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the KKS foundation, Sweden through the S.E.R.T. Research Profile project at Blekinge Institute of Technology. Robert Feldt has also been supported by the Swedish Scientific Council (No. 2015-04913, 'Basing Software Testing on Information Theory').

Data availability

Data will be made available on request.

References

Adamoli, A., Zaparanuks, D., Jovic, M., Hauswirth, M., 2011. Automated gui performance testing. Softw. Qual. J. 19 (4), 801–839.
Alegroth, E., Bache, G., Bache, E., 2015. On the industrial applicability of texttest: An empirical case study. In: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation. ICST, IEEE, pp. 1–10.
Alégroth, E., Feldt, R., 2017. On the long-term use of visual gui testing in industrial practice: a case study. Empir. Softw. Eng. 22 (6), 2937–2971.
Alegroth, E., Feldt, R., Olsson, H.H., 2013. Transitioning manual system test suites to automated testing: An industrial case study. In: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation. IEEE, pp. 56–65.
Alégroth, E., Gao, Z., Oliveira, R., Memon, A., 2015. Conceptualization and evaluation of component-based testing unified with visual gui testing: an empirical study. In: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation. ICST, IEEE, pp. 1–10.
Alégroth, E., Karlsson, A., Radway, A., 2018. Continuous integration and visual gui testing: Benefits and drawbacks in industrial practice. In: Software Testing, Verification and Validation (ICST), 2018 IEEE 11th International Conference on. IEEE, pp. 172–181.
Bertolino, A., Guerriero, A., Miranda, B., Pietrantuono, R., Russo, S., 2020. Learning-to-rank vs ranking-to-learn: Strategies for regression testing in continuous integration. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. ICSE '20, Association for Computing Machinery, New York, NY, USA, pp. 1–12. http://dx.doi.org/10.1145/3377811.3380369.
Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., Li, H., 2007. Learning to rank: from pairwise approach to listwise approach. In: Proceedings of the 24th International Conference on Machine Learning. pp. 129–136.
Chapelle, O., Metlzer, D., Zhang, Y., Grinspan, P., 2009. Expected reciprocal rank for graded relevance. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management. pp. 621–630.
Chen, T., He, T., Benesty, M., Khotilovich, V., 2019. Package 'xgboost'. p. 40, R version 90 (1-66).
Coppola, R., Ardito, L., Torchiano, M., 2019. Fragility of layout-based and visual gui test scripts: An assessment study on a hybrid mobile application. In: Proceedings of the 10th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation. A-TEST 2019, Association for Computing Machinery, New York, NY, USA, pp. 28–34. http://dx.doi.org/10.1145/3340433.3342824.

Coppola, R., Ardito, L., Torchiano, M., Alégroth, E., 2021. Translation from layout-based to visual android test scripts: An empirical evaluation. J. Syst. Softw. 171, 110845.
Coppola, R., Feldt, R., Nass, M., Alegroth, E., 2024. Ltr-similo replication package. http://dx.doi.org/10.6084/m9.figshare.26311990.
Coppola, R., Morisio, M., Torchiano, M., 2018. Mobile gui testing fragility: a study on open-source android applications. IEEE Trans. Reliab. 68 (1), 67–90.
Dobslaw, F., Feldt, R., Michaëlsson, D., Haar, P., de Oliveira Neto, F.G., Torkar, R., 2019. Estimating return on investment for gui test automation frameworks. In: 2019 IEEE 30th International Symposium on Software Reliability Engineering. ISSRE, IEEE, pp. 271–282.
Eladawy, H.M., Mohamed, A.E., Salem, S.A., 2018. A new algorithm for repairing web-locators using optimization techniques. In: 2018 13th International Conference on Computer Engineering and Systems. ICCES, IEEE, pp. 327–331.
Grechanik, M., Xie, Q., Fu, C., 2009a. Creating gui testing tools using accessibility technologies. In: Software Testing, Verification and Validation Workshops, 2009. ICSTW'09. International Conference on. IEEE, pp. 243–250.
Grechanik, M., Xie, Q., Fu, C., 2009b. Experimental assessment of manual versus tool-based maintenance of gui-directed test scripts. In: 2009 IEEE International Conference on Software Maintenance. IEEE, pp. 9–18.
Grechanik, M., Xie, Q., Fu, C., 2009c. Maintaining and evolving gui-directed test scripts. In: Proceedings of the 31st International Conference on Software Engineering. IEEE Computer Society, pp. 408–418.
Ibrahim, M., 2020. An empirical comparison of random forest-based and other learning-to-rank algorithms. Pattern Anal. Appl. 23 (3), 1133–1155.
Ishwaran, H., Kogalur, U.B., Kogalur, M.U.B., 2022. Package 'randomforestsrc'. Breast 6 (1).
Järvelin, K., Kekäläinen, J., 2017. Ir evaluation methods for retrieving highly relevant documents. In: ACM SIGIR Forum. Vol. 51, ACM New York, NY, USA, pp. 243–250.
Kaur, A., Kaur, I., 2018. An empirical evaluation of classification algorithms for fault prediction in open source projects. J. King Saud Univ.-Comput. Inf. Sci. 30 (1), 2–17.
Kirinuki, H., Tanno, H., Natsukawa, K., 2019. Color: Correct locator recommender for broken test scripts using various clues in web application. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering. SANER, Vol. 36, pp. 310–320, (4).
LaValley, M.P., 2008. Logistic regression. Circulation 117 (18), 2395–2399.
Leotta, M., Stocco, A., Ricca, F., Tonella, P., 2014. Reducing web test cases aging by means of robust xpath locators. In: 2014 IEEE International Symposium on Software Reliability Engineering Workshops. IEEE, pp. 449–454.
Leotta, M., Stocco, A., Ricca, F., Tonella, P., 2015. Using multi-locators to increase the robustness of web test cases. In: Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on. IEEE, pp. 1–10.
Leotta, M., Stocco, A., Ricca, F., Tonella, P., 2016. Robula+: An algorithm for generating robust xpath locators for web testing. J. Softw.: Evol. Process 28 (3), 177–204.
Liebel, G., Alégroth, E., Feldt, R., 2013. State-of-practice in gui-based system and acceptance testing: An industrial multiple-case study. In: 2013 39th Euromicro Conference on Software Engineering and Advanced Applications. IEEE, pp. 17–24.
Lin, C.-T., Yuan, S.-H., Intasara, J., 2021. A learning-to-rank based approach for improving regression test case prioritization. In: 2021 28th Asia-Pacific Software Engineering Conference. APSEC, IEEE, pp. 576–577.
Liu, T.-Y., et al., 2009. Learning to rank for information retrieval. Found. Trends® Inf. Retr. 3 (3), 225–331.
Mahmud, J., Cypher, A., Haber, E., Lau, T., 2014. Design and industrial evaluation of a tool supporting semi-automated website testing. Softw. Test. Verif. Reliab. 24 (1), 61–82.
Memon, A.M., Pollack, M.E., Soffa, M.L., 2001. Hierarchical gui test case generation using automated planning. IEEE Trans. Softw. Eng. 27 (2), 144–155.
Montoto, P., Pan, A., Raposo, J., Bellas, F., López, J., 2011. Automated browsing in ajax websites. Data Knowl. Eng. 70 (3), 269–283.
Moreira, R.M., Paiva, A.C., Nabuco, M., Memon, A., 2017. Pattern-based gui testing: Bridging the gap between design and quality assurance. Softw. Test. Verif. Reliab. 27 (3), e1629.
Nass, M., Alégroth, E., Feldt, R., 2021. Why many challenges with gui test automation (will) remain. Inf. Softw. Technol. 138, 106625.
Nass, M., Alégroth, E., Feldt, R., Coppola, R., 2023a. Robust web element identification for evolving applications by considering visual overlaps. In: 2023 IEEE Conference on Software Testing, Verification and Validation. ICST, IEEE, pp. 258–268.
Nass, M., Alégroth, E., Feldt, R., Leotta, M., Ricca, F., 2022. Similarity-based web element localization for robust test automation. ACM Trans. Softw. Eng. Methodol. 32 (3), 1–30. http://dx.doi.org/10.1145/3571855.
Nass, M., Alégroth, E., Feldt, R., Leotta, M., Ricca, F., 2023b. Similarity-based web element localization for robust test automation. ACM Trans. Softw. Eng. Methodol. 32 (3), 1–30.
Nielsen, D., 2016. Tree Boosting with Xgboost-Why Does Xgboost Win Every Machine Learning Competition? (Master's thesis). NTNU.
Olan, M., 2003. Unit testing: test early, test often. J. Comput. Sci. Coll. 19 (2), 319–328.
Omri, S., Sinz, C., 2022. Learning to rank for test case prioritization. In: 2022 IEEE/ACM 15th International Workshop on Search-Based Software Testing. SBST, IEEE, pp. 16–24.
Online, 2022a. Css selectors. URL https://en.wikipedia.org/wiki/CSS.
Online, 2022b. Dom. URL https://www.w3.org/TR/WD-DOM/introduction.html.
Online, 2022c. Xpath. URL https://en.wikipedia.org/wiki/XPath.
Palo, H.K., Sahoo, S., Subudhi, A.K., 2021. Dimensionality reduction techniques: Principles, benefits, and limitations. In: Data Analytics in Bioinformatics: A Machine Learning Perspective. pp. 77–107.
Ricca, F., Leotta, M., Stocco, A., 2019. Three open problems in the context of e2e web testing and a vision: Neonate. In: Advances in Computers. Vol. 113, Elsevier, pp. 89–133.
Rigatti, S.J., 2017. Random forest. J. Insur. Med. 47 (1), 31–39.
Safdari, N., Alrubaye, H., Aljedaani, W., Baez, B.B., DiStasi, A., Mkaouer, M.W., 2019. Learning to rank faulty source files for dependent bug reports. In: Big Data: Learning, Analytics, and Applications. Vol. 10989, SPIE, pp. 60–78.
Tian, Y., Wijedasa, D., Lo, D., Le Goues, C., 2016. Learning to rank for bug report assignee recommendation. In: 2016 IEEE 24th International Conference on Program Comprehension. ICPC, IEEE, pp. 1–10.
Tonella, P., Ricca, F., Marchetto, A., 2014. Recent advances in web testing. In: Advances in Computers. Vol. 93, Elsevier, pp. 1–51.
Wood, L., Le Hors, A., Apparao, V., Byrne, S., Champion, M., Isaacs, S., Jacobs, I., Nicol, G., Robie, J., Sutor, R., et al., 1998. Document object model (dom) level 1 specification. W3C Recomm. 1.
Yang, X., Tang, K., Yao, X., 2014. A learning-to-rank approach to software defect prediction. IEEE Trans. Reliab. 64 (1), 234–246.
Yaraghi, A.S., Bagherzadeh, M., Kahani, N., Briand, L., 2022. Scalable and accurate test case prioritization in continuous integration contexts. IEEE Trans. Softw. Eng. 1–24. http://dx.doi.org/10.1109/TSE.2022.3184842.
Yeh, T., Chang, T.-H., Miller, R.C., 2009. Sikuli: using gui screenshots for search and automation. In: Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology. pp. 183–192.
Yu, X., Dai, H., Li, L., Gu, X., Keung, J.W., Bennin, K.E., Li, F., Liu, J., 2023. Finding the best learning to rank algorithms for effort-aware defect prediction. Inf. Softw. Technol. 157, 107165.

Riccardo Coppola received his Ph.D. in Automation and Computer Engineering at the Polytechnic University of Turin, where he currently serves as Assistant Professor in the SoftEng research group. His research interests include automated GUI testing for web and mobile applications, the evaluation of nonfunctional properties of software, and the application of gamification to Software Engineering practices.

Robert Feldt is professor of Software Engineering at Chalmers University of Technology and Mid Sweden University. He is passionate about a wide range of topics from human factors and automation to statistics, causal and bayesian analysis, and applied machine learning (ML). His primary focus is on software testing and quality, along with humancentered software engineering. Lately, he is also been working on using ML in medicine/healthcare and materials science. He often collaborates with companies in Sweden, Europe, and Asia, while also leading more basic research. In 2002, he earned a Ph.D. in Computer Engineering from Chalmers University of Technology. He also studied Psychology at Gothenburg University in the 1990s. Beyond academia, he brings over 30 years of experience as an IT, software, and AI/ML consultant. In recent years, he has worked as an AI strategist helping upper management learn about AI and change their organizations through AI/ML-based improvement projects. Dr. Feldt contributes as the coEditor in Chief of the Empirical Software Engineering journal and serves on the editorial boards of two other journals, STVR (software testing) and SQJ (software quality). He has published more than 165 peer-reviewed, scientific papers and won several best paper awards. He has a patent on a method for testing and prioritizing inputs to Deep Neural Nets.

Michel Nass (Ph.D.): Dr. Nass is, at the time of writing, a project assistant at Blekinge Institute of Technology. He successfully defended his dissertation " On overcoming challenges with GUI-based test automation" in 2024. His research focus is within the area of GUI-based testing with special interest in technical solutions such as tools and frameworks to support efficient and effective testing in industrial practice. In addition to his academic career, Dr. Nass has been active for over 30 years as an industrial consultant in software engineering, specialized in testing, verification and validation.

Emil Alégroth (Ph.D.): Dr. Alégroth is a senior lecturer at Blekinge Institute of Technology. His research focus is on empirical software engineering research in close collaboration with industry. His research focus in the area of automated GUI testing, but has also conducted and published research in the related areas of model-based testing, software compliance, quality assurance, security testing, test methods, GenAI-based testing and more. In addition to his research background, Dr. Alégroth has almost 10 years of industrial experience as an expert QA consultant in industry.