Search-Based Robust XPath Generation

CHANYOUNG RYU, KAIST Department of Computer Science
CHANMIN PARK, KAIST Department of Computer Science
JEONGHO HA, KAIST Department of Computer Science
SANGWOO HAHN, KAIST Department of Computer Science

1 INTRODUCTION

In web design, developers need the ability to make quick updates to websites while retaining prior functionality. A key part of the development process is automated regression testing, in which a website's functionality is verified to work despite changes. Elements of a website are found using XML Paths (XPaths), which locates elements by representing them with relative or absolute paths, similar to those used in file systems. XPaths are reliable in locating elements in static websites, but as a website's structure changes, they may not be able to locate the same elements. Therefore, a key problem in automated web regression testing is to find XPaths that can survive version changes, also known as robust XPaths.

A greedy algorithm approach is able to generate optimal robust XPaths by searching through the entire set of possible XPaths. However, this set increases exponentially with the number of attributes in an element and the number of elements in a webpage, deeming this approach impractical. Other search-based approaches have been proposed to generate less-than-optimal XPaths but no empirical tests have verified the validity of these claims.

In this paper, we propose a genetic algorithm to generate robust XPaths. Among other search-based methods, a genetic algorithm is a good fit as mutation and crossover operators are clearly defined and enable the exploration of the entire search space. (INSERT BRIEF EVALUATION CONCLUSIONS HERE)

An overview of the paper goes as following: in Section II, background information is provided and the terminology used in the paper is defined; in Section III, previous works and their limitations are outlined; in Section IV, components of the genetic algorithm, such as selection methods and exploration operators, are described in detail; in Section V, the dataset used for evaluation is described; in Section VI, the performance of our algorithm is evaluated against a commercial XPath generator; and in Section VII, potential future works are introduced.

2 BACKGROUND

Web pages are represented as document object model (DOM) trees, where each element is a node in the tree. Developers must find a way to represent and locate these elements to perform automatic testing. Among various methods such as regular expressions and CSS selectors, XPaths exploit the tree structure to represent elements in an intuitive yet precise manner. They allow for a relative search anywhere in the document for elements that result in a match. For example, "//html/div[@class='X'][1]/p[@id='Y']" represents finds all 'p' tag elements with an ID value of 'Y', under the first 'div' tag with class name 'X' under its parent, under an 'html' tag. For the purposes of this paper, an XPath consists only of tags and attributes, where attributes include class name, ID

Authors' addresses: Chanyoung RyuKAIST Department of Computer Science; Chanmin Park, cmpark0917@kaist.ac.krKAIST Department of Computer Science; JeongHo Ha, hajh@kaist.ac.krKAIST Department of Computer Science; Sangwoo Hahn, thahn106@kaist.ac.krKAIST Department of Computer Science.

value, and position under its parent.

When an XPath uniquely identifies an element in the web page, it is a locator for that element. For static websites, any XPath that is a locator can be used for testing. However, modern websites undergo frequent updates that may change the DOM tree. This implies that an XPath must also be able to locate the same element in future versions of a website for effective regression testing. Otherwise, a new XPath must be generated and validated, adding an entire layer of uncertainty. Therefore, generating robust XPaths, those that are able to locate the same element despite changes to the website structure, is an important task in web development.

To evaluate the robustness of an XPath, a metric called the fragility coefficient is defined to measure the fragility (inversely proportional to robustness) of an XPath. Intuitively, it would be the case that certain parts of an XPath make it more fragile; for example, adding a position attribute to an XPath will make it more fragile, since inserting or removing an element above the target element likely prevents the XPath from locating the same element. Thus, a weight proportional to fragility is associated with each part of an XPath, and these weights are summed to generate a fragility coefficient for an XPath. These weights are not defined, so a significant part of the evaluation was dedicated to determine them. Further details are provided in Section VI.

3 METHODOLOGY

This section provides a detailed look at the formulation of the genetic algorithm. Chromosome: a chromosome is an XPath.
Population: a population is a set of chromosomes; that is, a set of XPaths. Furthermore, two sub-populations are distinguished since separate fitness functions must be used for each:

- pop': the set of XPaths that are not locators of the target element; that is, it locates two or more elements in the DOM tree.
- pop": the set of XPaths that are locators of the target element.

The initial population is generated with following three types of XPaths:

(1) "//*" – the most generic XPath
(2) The full absolute XPath – the XPath containing every attribute from every level to the target element
(3) XPaths generated by removing highest layer of the full absolute XPath one at a time

Fitness function: the fitness is defined separately for the two sub-populations:

- pop': number of elements located by the XPath.
- pop": the fragility coefficient described in Section VI.

It is important to note that when comparing an element from pop' and an element from pop", the latter has a lower fitness score; that is, XPaths that are locators are always preferred over non-locators.
Parent selection: tournament selection with a tournament size of 3 is used to select parents. Tournament selection is preferred over fitness proportional selection or ranking selection due to its balance between exploration and exploitation when applying evolutionary pressure.
Generational selection: modified generational replacement is used to select children, the modification being that the initial population is always retained in every generation. Since the initial population exhibits great diversity, from the most generic XPath to the most specific locator, it is maintained in every generation to guarantee

diversity. Elitism is an alternative to generation replacement. However, when keeping the select best from the parents, XPaths in pop" would always be ranked higher than pop'. Thus, those in pop' would not be able to survive many generations, significantly reducing the diversity of the population.
Mutation: a total six mutations, three pairs of two complementary mutations, are defined. The set of mutations are described in (INSERT TABLE NAME).
Crossover: only single-point crossover is defined. Given two parent chromosomes, this crossover method randomly chooses a level from the bottom, splits the chromosomes, and recombines them. An example of a crossover operation is shown in (INSERT FIGURE NAME). Parameters: the parameters are set as shown in (INSERT FIGURE NAME).

4 DATASET

4.1 Test Subjects

The test subjects of our XPath locator are html files of websites since the locator is DOM-based. We have devised some conditions that must be satisfied to work as a test dataset of this XPath locator. The conditions are as follows:

(1) It should have both the old and new version of the website, which has undergone a structural change during the time gap.
(2) It should have a same element, in both the old and new version of the website.

An element could range from text of an article/post, to image or video files. However, in our test dataset, selected target elements used are texts. This is because text elements in the DOM are generally more concise and shorter than image elements.

4.2 Data Collection

The test datasets are collected from <Internet Archive Wayback Machine> . We collected a total of 18 test sets of various types. The types include news articles, social network services, enterprise's official website…etc. We divided the test sets into two types, which are as follows:

- *Type 1:* Google chrome XPath from old version can locate designated element in the new version.
- *Type 2:* Google chrome XPath from old version cannot locate designated element in the new version.

This we because we wanted to check if our locator is at least as good as Google chrome locator by passing all test cases from type 1, and hopefully show better performance by passing some of the test cases from type 2. Of the 18 test datasets, 5 of them are type 1 and the other 13 are type 2. The 2016 Facebook main page is a special test case because it contains both type 1 and type 2 elements.

5 EVALUATION

5.1 Parameters

As the reference paper only supplied an approach for the genetic algorithm rather than the specifications, it was necessary to choose the parameters for the genetic algorithm and the fitness function.

5.2 Coverage

Our algorithm was able to cover a higher rate of

Table 1. Mutation Operations

trans_add_name	replaces the * in the highest level to a specific tag
trans_remove_name	replaces a specific tag in the highest level to a *
trans_add_predicate	adds a predicate to the highest level
trans_remove_predicate	removes a predicate at the highest level

Source: This is a table sourcenote. This is a table sourcenote. This is a table sourcenote. *Note:* This is a table footnote.

5.3 Time Costs

6 FUTURE WORK

While the empirical evaluation showed promising results, the scope of the analysis was very limited as only 18 target elements in total were tested.