Automated Generation of Metamorphic Relations for Query-Based Systems

Sergio Segura
sergiosegura@us.es
SCORE Lab, I3US Institute,
Universidad de Sevilla
Seville, Spain

Juan C. Alonso
javalenzuela@us.es
SCORE Lab, I3US Institute,
Universidad de Sevilla
Seville, Spain

Alberto Martin-Lopez
alberto.martin@us.es
SCORE Lab, I3US Institute,
Universidad de Sevilla
Seville, Spain

Amador Durán
amador@us.es
SCORE Lab, I3US Institute,
Universidad de Sevilla
Seville, Spain

Javier Troya
jtroya@uma.es
ITIS Software, Universidad de Málaga
Malaga, Spain

Antonio Ruiz-Cortés
aruiz@us.es
SCORE Lab, I3US Institute,
Universidad de Sevilla
Seville, Spain

ABSTRACT

Searching and displaying data based on user queries is a pervasive feature of most software applications such as information systems, web portals, and web APIs. The large volume of data managed by these types of systems, henceforth called *query-based systems* (QBSs), makes them extremely hard to test due to the difficulty to assess whether the output of a query is correct, the so-called *test oracle problem*. Metamorphic testing has proved to be a very effective approach to alleviate the oracle problem in QBSs, by exploiting the relations among multiple executions of the QBS under test, so-called *metamorphic relations* (MRs). However, the identification of MRs mostly remains a manual and creative task, limiting the applicability of the approach. In this paper, we propose a method for the automated generation of MRs in QBSs starting from a lightweight specification of the query parameters of the system. Evaluation results show that hundreds of MRs can be automatically identified in real-world systems like IMDb, SkyScanner, or YouTube in just a few seconds.

CCS CONCEPTS

• **Software and its engineering** → **Software testing**.

KEYWORDS

Metamorphic testing, metamorphic relation, oracle problem.

ACM Reference Format:
Sergio Segura, Juan C. Alonso, Alberto Martin-Lopez, Amador Durán, Javier Troya, and Antonio Ruiz-Cortés. 2022. Automated Generation of Metamorphic Relations for Query-Based Systems. In *7th International Workshop on Metamorphic Testing (MET'22), May 9, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3524846.3527338

1 INTRODUCTION

Most software applications support searching and displaying data based on user queries. A *query* specifies user preferences—the data to be retrieved and how to display it—declaratively using visual forms or textual languages, which typically support standard operations such as filtering, ordering or pagination. In what follows, we refer to software systems supporting user queries as *query-based systems* (QBS). Typical examples of QBS are information systems like OpenERP (supporting queries like "retrieve invoices issued before 2022"), e-commerce sites like eBay (e.g., "search for microphones under $100"), software project platforms like GitHub (e.g., "get Node.js projects with less than 10 committers"), video streaming apps like Netflix (e.g., "order series by popularity"), email clients like Gmail (e.g., "display messages including the word *conference* in the subject"), or even video games like World of Warcraft (e.g., "get progression data for character *Thrall* in *Blackrock* realm"). Testing QBS is extremely challenging as they suffer from the oracle problem: it is very difficult, often infeasible, to assess whether the output of a query is correct, either because the expected output is unknown or because it is hard to compare it to the observed output [7, 28].

Metamorphic testing (MT) [9, 25] alleviates the oracle problem by adopting an innovative approach to testing: instead of checking the correctness of each individual output, MT looks at the relations among the input and output of two or more executions of the program under test, known as *metamorphic relations* (MRs). MT has been successfully applied to detect failures in different types of QBS including search engines such as Google and Bing [34–36], RESTful APIs such as those of Spotify and YouTube [24], e-commerce sites such as Amazon and Walmart [33], and data repositories like the NASA's Data Access Toolkit [15]. MRs in QBS specify the relation between the inputs and outputs of multiple queries. For example, the following is a simple metamorphic relation for video-sharing platforms like YouTube and Vimeo: the number of videos returned in a search for the keyword K should be equal or higher than the number of videos returned in a search for K using the `High-Definition` filter.

MRs can be often defined at a very abstract level, representing not a single relation, but a set of MRs. Inspired by this idea, the concept of metamorphic relation patterns has been exploited by different groups of authors [21, 24, 33]. Zhou et al. [33] define a *metamorphic relation pattern* as an abstraction that characterizes a set of (possibly infinitely many) MRs. MR patterns have proved to be very helpful in guiding testers on the search for MRs with a certain structure, making the identification of the relations significantly easier than when starting from scratch. In a previous paper [23], we presented a catalogue of seven MR patterns to assist testers in the identification of MRs in QBSs. The "disjoint partitions" pattern, for example, represents those relations where the outputs of the test cases should be pairwise disjoint (i.e., they should have no items in common) because the underlying relation can be partitioned according to the values of at least one input attribute. In YouTube, for example, two searches with the same keywords for videos in two dimensions (2D) and three dimensions (3D) should return disjoint results, since the same video cannot be 2D and 3D at the same time. A full description of MR patterns for QBSs is presented in Section 2.

MR patterns have proved to be very helpful in guiding the identification of MRs in QBSs, however, the identification of the MRs is still mostly a manual task that requires time and creativity [25]. Automated approaches for the identification of MRs typically target very specific domains (e.g., [27, 32]), generate false positives (e.g., [8, 14, 32]), and require access to the source code [13, 14, 27], input/output datasets [6, 32], or existing MRs [18, 29] of the program under test. This limits the applicability of the approach and the degree of automation that can be achieved [25].

In this paper, we present an approach for the automated identification of MRs in QBSs by leveraging MR patterns. Starting from a lightweight specification of the query parameters and a given test suite, our tool is able to automatically identify instances of the proposed MR patterns for QBSs. Evaluation results with 8 real-world web applications and APIs show that our approach can efficiently identify hundreds of MRs in QBSs such as YouTube, IMDb, and SkyScanner in a few seconds. In contrast to existing approaches, no access to the source code or previous datasets of input and output values are required. More importantly, no false positives are generated, as long as the input specification is correct. These results set the path for new and exciting research directions on the generation of MRs driven by the specification and MR patterns.

The paper is structured as follows. Section 2 describes the MR patterns for QBSs that form the basis of our work. Our approach and the companion tool, MRG, are presented in Section 3. Experimental results are reported in Section 4. Limitations, open challenges, threats to validity, and related work are discussed in Sections 5, 6, 7, and 8, respectively. Finally, Section 9 concludes the paper.

2 METAMORPHIC RELATION PATTERNS FOR QBSS

In previous work [23], we presented a catalogue of MR patterns for QBSs. These patterns are the basis of our proposal. In what follows, we present a brief summary of each pattern and refer the reader to the original paper for further details.

Input equivalence. This pattern represents those relations where the source and the follow-up test cases are equivalent and therefore their outputs should contain the same items in the same order, i.e., they should be equal. There exist two typical situations where this pattern can be instantiated. The first one appears when input parameters accept equivalent values expressed in different formats, e.g., 1MB ≡ 1024KB. The second situation appears in most queries and it is related to default values: it should not matter whether or not default values of the query parameters are specified. For example, a search in YouTube specifying no order should produce exactly the same result as indicating the default ordering criterion, which is based on the *relevance* to the search query. Lindvall et al. [15] used a rich set of these types of metamorphic relations to reveal failures in the NASA DAT, a large database of telemetry data.

Shuffling. This pattern represents those metamorphic relations where the source and follow–up outputs should contain the same items regardless of the ordering criteria specified as input. For example, a search for "*hotels in London*" in Booking.com should return the same results regardless of the ordering criteria specified (price, review score, stars, etc.).

Conjunctive conditions. This pattern groups those relations where the query is iteratively refined adding new conjunctive conditions such that the results of each test case should be a subset of the results of the previous ones. This pattern is very common in query operations where most of the parameters are filters. For example, suppose we perform a search for YouTube videos of "pets". Next, we search for videos of "pets" in three dimension (3D), and finally we search for videos of "pets" in 3D uploaded after 2018. Intuitively, the results of the third search (videos of pets in 3D published after 2018) should be a subset of the result set of the second search (videos of "pets" in 3D), and in turn these should be a subset of the results of the original search (videos of "pets"). This pattern, and slight variations of it, were extensively used by Zhou et al. in their papers on testing search engines [34, 36], and also by Segura et al. in their work on testing RESTful web APIs [24].

Disjunctive conditions. This pattern is similar to the previous one, but the query is expanded with input disjunctive conditions such that the results of each test case should be a subset of the following ones. For example, let us suppose a search in IEEE Xplore for papers including the word "metamorphic" in their title. Next, we expand the search to papers including either "metamorphic" OR "testing". Naturally, the papers returned in the former search should be a subset of those found in the second search. This pattern has been previously exploited in the context of search engines [34, 36].

Disjoint partitions. This pattern represents those relations where the outputs of the follow–up test cases should be pairwise disjoint (i.e., they should have no items in common) because the underlying relation can be partitioned according to the values of at least one input attribute. For instance, suppose a search in a PayPal user's account for refunds with status *COMPLETED*. Next, let us suppose a new search is performed in the same account for refunds with status *CANCELLED*. The results set of both searches should have no items in common.

Complete partitions. This pattern is related to the previous one, and it represents those relations where the union of the follow–up outputs should contain the same items as the source output because the underlying relation can be partitioned according to the values of at least one input attribute. For instance, YouTube videos are classified according to its duration in *short* (less than 4 minutes), *medium* (between 4 and 20 minutes) and *long* videos (longer than 20 minutes). Hence, a YouTube search for videos of "testing" should return the same videos as the joint result set of three searches for *short*, *medium*, and *long* videos of "testing".

3 AUTOMATED GENERATION OF MRS

In this section, we describe the proposed method for the automated generation of MRs in QBSs and the tool implementing it. Roughly speaking, our work is based on the observation that, given a QBS under test and an MR pattern, the applicability of the pattern is typically related to the existence of a certain type of parameter in the QBS. For example, the *conjunctive conditions* pattern is typically applicable whenever a QBS provides *filter* parameters for restricting searches. Hence, we first propose classifying input parameters based on their potential to create instances of the target patterns. Then, we propose creating MRs by systematically adding, removing, and replacing input parameters and their values from the source test cases.

Our method receives two inputs: 1) an input specification of the QBS under test, and 2) a set of existing test cases revealing no failures (i.e., source test cases). The input specification must include, among others, a classification of the query parameters, as described next, based on their potential to create MRs derived from the patterns described in Section 2. Then, a set of transformations are applied to the input source test cases generating the output MRs. Those MRs can then be used by testers for the creation of specific metamorphic test cases, either manually or automatically, by assigning specific test values to input parameters.

In what follows, we describe the proposed classification of query parameters, the input specification required by our approach, the proposed transformations for the generation of MRs, and the tool support.

3.1 Classification of query parameters

In this section, we present a classification of query parameters using examples and explaining how they can be used to identify instances of the patterns presented in Section 2.

Conjunctive filters. These are parameters that allow to restrict the results of a query. For example, the `Category` parameter in Vimeo restricts the search to those videos in a certain category, e.g., Comedy. This type of parameters are extremely common in QBSs and they can be used, in isolation or together with similar parameters, to identify instances of the *conjunctive conditions* MR pattern. For example, we could perform a search in Vimeo for videos of 'MET' and then repeat the search restricting it to videos of the `Category` 'Talks' `uploaded` in the 'last 7 days'. The results of the second search should be a subset of those returned in the first search.

Disjunctive filters. These are parameters that allow to expand the results of a query. For example, when searching flights in SkyScanner the user can submit the parameter `Add nearby airports` expanding the search beyond the origin and destination airports indicated in the form. Analogously to conjunctive filters, this type of parameters can be used to identify instances of the *disjunctive conditions* MR pattern.

Disjoint filters. These are parameters with a discrete set of values that allow to split the results of a query into disjoint partitions. For example, the `Price` parameter in Vimeo makes it possible to restrict searches to 'Free' or 'Paid' videos. Whenever this type of parameter appears, it is possible to identify instances of the *disjoint partitions* MR pattern, e.g. the results of two searches with the same query for 'Free' and 'Paid' videos should have no videos in common.

Complete filters. These are parameters with a discrete set of values that allow to split the results of a query into complete partitions (not necessarily disjoint). For example, the `Meals` parameter in TripAdvisor.com allows to restrict the searches for restaurants to those serving 'Breakfast', 'Brunch', 'Lunch', and/or 'Dinner'. This type of parameters enable the generation of instances of the *complete partitions* MR pattern, e.g., a search for restaurants in 'Rome' should return the same items as the joint results of four searches for restaurants in 'Rome' serving 'Breakfast', 'Brunch', 'Lunch', and 'Dinner'.

Default values. These are optional parameters with explicit default values. In the Amadeus API, for example, the default number of flight offers to return (parameter `max`) is 50. Therefore, calling the API without the parameter `max` is equivalent to setting it to its default value (50). This kind of parameters can be used to find instances of the *input equivalence* MR pattern.

Ordering parameters. These are parameters used to specify the order in which the results will be displayed. Whenever one or more ordering parameters exist, it is possible to find instances of the *shuffling* MR pattern, as long as the number of results is not limited.

Limit parameters. These are parameters that limit the number of results to be returned by the query. In contrast to the other types of parameters, limit parameters are not used to generate MRs. On the contrary, their inclusion in a test case restricts the applicability of some of the other parameters for creating MRs and thus they must be defined explicitly. For example, two searches in YouTube with the same keyword and different ordering criteria will probably return different videos if the number of results is limited to 10 (parameter `maxResults`), making not applicable the *shuffling* MR pattern.

Note that the previous classification is not complete and so some parameters may not fall within any of the categories. Similarly, a parameter can belong to more than one category and therefore it can be used for the identification of different types of MRs.

3.2 Input specification

Our approach for the identification of MRs requires an input specification of the QBS under test. As an example, Figure 1 depicts the input specification of the operation `GET incidents` in the real-world Bikewise web API[1] [1]. This operation allows users to search bicycle incidents by date and location. For the sake of generalizability, we used a custom domain-independent specification format using Yet Another Markup Language (YAML) notation and mostly based on the OpenAPI Specification (OAS) language for web APIs [3]. The input specification should describe the following items:

(1) *Parameters.* For each parameter, the specification should include, at least, its name, type (or possible values), default value (if any), and whether the parameter is required or not.

[1]Some minor changes have been introduced with respect to the real API for illustrative purposes.

The specification in Figure 1 describes nine parameters (lines 6-56).

(2) *Constraints*. The specification should include the dependency constraints among the input parameters such as "A requires B" or "A excludes B". Inter-parameter dependencies are very common in QBSs and it is hardly possible to generate valid test cases without defining them explicitly [20]. In the Bikewise example, one dependency constraint is specified (lines 59-60).

(3) *Query parameters*. This is the part specific to our approach (lines 63-82) and it requires each query parameter to be classified into the categories described in the previous section, excluding default values, which are typically included in the specification of each parameter (e.g., line 43). In the example, there are one ordering parameter, four conjunctive filters, one disjoint filter, and one complete filter. Notice that the parameter `incident_type` falls into three different categories.

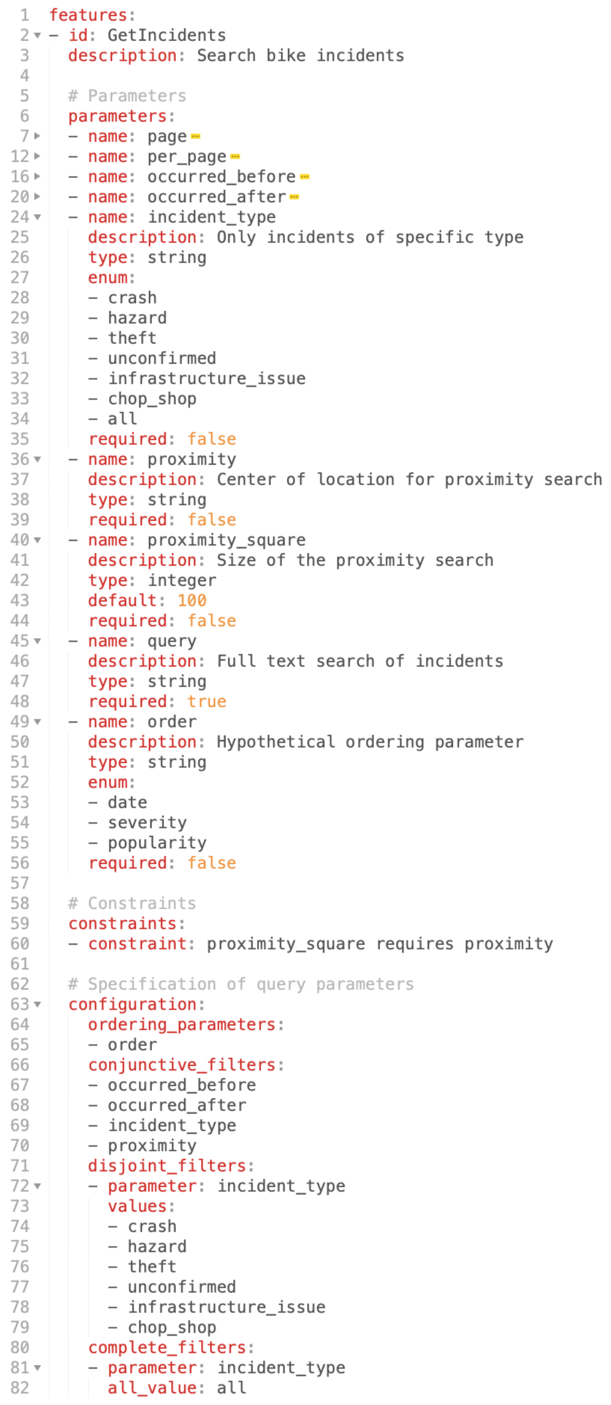
It is worth noting that the information related to (1) and (2) is typically provided in (domain-dependent) standard specification languages like OAS [3] or RAML [5]. Also, this data is typically required as input for the generation of combinatorial test cases in tools like ACTS [31] and PICT [4], and so in this regard our approach does not place a new burden in the user. Hence, we remark that our approach could work with any standard specification language including the data described above—points (1) and (2)—plus a lightweight classification of the query parameters based on the categories described in Section 3.1—point (3).

Regarding the classification of query parameters, it may optionally include the specific values that allow a disjoint filter to split the results of the query into disjoint partitions (property `values`). In Figure 1, all the possible values of the parameter `incident_type` are selected except `'all'`. Analogously, complete filters must include an additional property (`all_value`) indicating the input value required to return all the possible matches. In the example, this value is `'all'`, but in other parameters and QBSs it could take a different name (in YouTube this value is typically `'any'`) or `'empty'`, indicating that all items are returned by default when the parameter is not provided.

3.3 Generation of metamorphic relations

We propose to identify MRs by applying *input transformations* to existing test cases revealing no failures, i.e., source test cases. Specifically, each transformation creates one or more follow-up test cases by systematically adding, removing or replacing parameters and their values from the source test case. Each type of input transformation aims to find instances of a specific MR pattern and relies on the information provided in the input specification of the QBS under test.

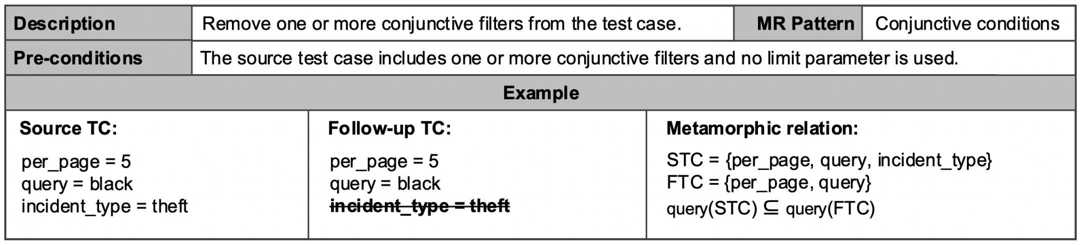
Figure 2 describes one of the input transformations used in our work. The transformation tries to find instances of the *conjunctive conditions* MR pattern by removing one or more conjunctive filters from the source test case (if any). In the example, a follow-up test case is created by removing the filter `incident_type` from the source test case. As a result, a MR is created including the source test case and the follow-up test case derived from it, plus an output relation (oracle), which is specific for each type of input transformation.



```
1   features:
2   - id: GetIncidents
3     description: Search bike incidents
4
5     # Parameters
6     parameters:
7     - name: page
12    - name: per_page
16    - name: occurred_before
20    - name: occurred_after
24    - name: incident_type
25      description: Only incidents of specific type
26      type: string
27      enum:
28      - crash
29      - hazard
30      - theft
31      - unconfirmed
32      - infrastructure_issue
33      - chop_shop
34      - all
35      required: false
36    - name: proximity
37      description: Center of location for proximity search
38      type: string
39      required: false
40    - name: proximity_square
41      description: Size of the proximity search
42      type: integer
43      default: 100
44      required: false
45    - name: query
46      description: Full text search of incidents
47      type: string
48      required: true
49    - name: order
50      description: Hypothetical ordering parameter
51      type: string
52      enum:
53      - date
54      - severity
55      - popularity
56      required: false
57
58    # Constraints
59    constraints:
60    - constraint: proximity_square requires proximity
61
62    # Specification of query parameters
63    configuration:
64      ordering_parameters:
65      - order
66      conjunctive_filters:
67      - occurred_before
68      - occurred_after
69      - incident_type
70      - proximity
71      disjoint_filters:
72      - parameter: incident_type
73        values:
74        - crash
75        - hazard
76        - theft
77        - unconfirmed
78        - infrastructure_issue
79        - chop_shop
80      complete_filters:
81      - parameter: incident_type
82        all_value: all
```

Figure 1: Bikewise API input specification.

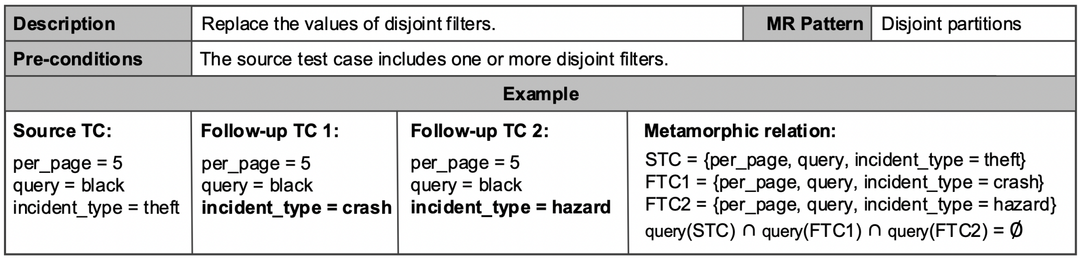
Notice that the values of the parameters that have no effect in the output relation are left undefined in the MR, making it reusable (a key property of MRs). This makes it possible to instantiate each MR into multiple specific metamorphic tests, e.g., by assigning different values to the parameters `per_page`, `query`, and `incident_type` in the example. Also, note that the relation is presented succinctly for



Description	Remove one or more conjunctive filters from the test case.	**MR Pattern**	Conjunctive conditions
Pre-conditions	The source test case includes one or more conjunctive filters and no limit parameter is used.		
Example			

Source TC:
per_page = 5
query = black
incident_type = theft
	Follow-up TC:
	per_page = 5
	query = black
	~~incident_type = theft~~
		Metamorphic relation:
		STC = {per_page, query, incident_type}
		FTC = {per_page, query}
		query(STC) $\subseteq$ query(FTC)	

Figure 2: Input transformation removing conjunctive filters.



Description	Replace the values of disjoint filters.	**MR Pattern**	Disjoint partitions
Pre-conditions	The source test case includes one or more disjoint filters.		
Example			

Source TC:
per_page = 5
query = black
incident_type = theft
	Follow-up TC 1:
	per_page = 5
	query = black
	incident_type = crash
		Follow-up TC 2:
		per_page = 5
		query = black
		incident_type = hazard
			Metamorphic relation:
			STC = {per_page, query, incident_type = theft}
			FTC1 = {per_page, query, incident_type = crash}
			FTC2 = {per_page, query, incident_type = hazard}
			query(STC) $\cap$ query(FTC1) $\cap$ query(FTC2) = $\emptyset$

Figure 3: Input transformation replacing the values of disjoint filters.

readability, but it can be automatically translated to a user-friendly version using natural language (see Section 4.2).

Figure 3 depicts another type of input transformation. The transformation aims to find instances of the *disjoint partitions* pattern by replacing the value of a disjoint filter. In the example, two follow-up test cases are created by replacing the value of the disjoint filter `incident_type` by `'crash'`, and `'hazard'`, respectively. The resulting MR is composed of a source test case, two follow-up test cases, and the corresponding output relation, i.e., the result sets of the three test cases should have no items in common.

Each type of input transformation may require the source test case to meet some pre-conditions like including or excluding a certain type of parameter (see examples in Figures 2 and 3). Similarly, some transformations could violate the dependencies among input parameters (defined in the input specification) and this must be checked to avoid invalid test cases. Finally, notice that input transformations can be applied in different ways creating different MRs. For example, we could remove `n` parameters at once in a single follow-up test case, or we could remove one parameter at a time along `n` different follow-up test cases.

3.4 Tool support

Our approach is supported by a Java tool, *MRG (Metamorphic Relation Generator)*, for the automated generation of MRs for QBSs [2]. The tool receives two inputs, namely: 1) an input specification of the QBS under test in YAML notation (see example in Figure 1), and 2) a test suite in comma separated format (CSV), where each test case is a list of pairs *<parameter,value>*. For each input test case, the tool tries to identify random instances of each MR pattern by applying nine different input transformations, as described in Section 3.3. Different stopping criteria can be configured, including a maximum number of iterations or a specific number of MRs to be generated. Two MRs are considered as duplicates if they use the same source test case and apply exactly the same changes (adding, removing, modifying input parameters) on each follow-up test case. Duplicated MRs are automatically discarded. Output MRs can be written in a machine-readable format (CSV) or in natural language following the MR template proposed by Segura et al. [22], making them homogeneous and easier to understand by users.

4 EVALUATION

We aim to study the efficacy and the effectiveness of our approach in identifying MRs in real-world QBSs, including the number and shape of the identified relations, as well as the performance of the proposed method. In what follows, we describe the setup and the results of the experiment.

4.1 Experimental setup

We assessed the effectiveness of MRG at identifying MRs in 8 real-world QBSs, including both web applications and web APIs from different domains, listed in Table 1. For each subject QBS, the table shows the specific use case, number of parameters, number of constraints, and number of each type of query parameter according to the classification presented in Section 3.1.

For each QBS, we wrote an input specification in YAML notation as described above and we generated a test suite (source test cases) using the Pairwise Independent Combinatorial Tool (PICT), by Microsoft [4]. Then, we ran MRG with both inputs to identify as many random MRs as possible. As the stopping criterion, we set as the maximum number of iterations for each input transformation the number of query parameters in the QBS potentially applicable to the transformation. Hence, for example, if the system under test has 5 conjunctive filters, we tried to generate instances of the *conjunctive conditions* MR pattern 5 times at most for each source test case. The generation of MRs was repeated 10 times to mitigate the effect of randomness. During our preliminary tests, we found

Name	Use case	Parameters	Constraints	MT Parameter Specification						
				Conjunctive	Disjunctive	Disjoint	Complete	Ordering	Defaults	Limit
IMDb	Title search	32	0	27	0	2	7	1	6	0
iTunes	Search	10	48	4	0	3	4	0	6	1
Kickstarter	Discover projects	14	157	13	0	9	7	1	2	0
Prestashop	List orders	11	0	9	0	2	2	1	1	0
SkyScanner	Flight search	14	15	1	2	0	0	0	4	0
Steam	Search store	14	0	13	0	1	1	1	1	0
Transfermarkt	Personal data search	16	0	15	0	4	1	0	0	1
YouTube	Search	31	56	24	0	8	5	1	16	1

Table 1: Subject query-based systems.

Name	Source test cases	Metamorphic relations							Follow-up test cases			Time (s)
		CC	DC	DP	CP	S	IE	Total	Min	Max	Avg	
IMDb	20	1017.8	0	29.1	6	20	0	1072.9	1	6	3.19	41.54
iTunes	15	0	0	18.7	0	0	45.8	64.5	1	6	2.27	0.09
Kickstarter	18	332.2	0	106.6	42.2	18	0	499	1	6	2.56	2.89
Prestashop	18	232.3	0	11	7	36	0	286.3	1	6	2.32	0.62
SkyScanner	12	6	10.9	0	0	0	0	16.9	1	2	1.21	0.03
Steam	17	372.1	0	11	0	29	5	417.1	1	6	2.70	1.55
Transfermarkt	20	0	0	72	0	0	0	72	1	6	3.55	0.12
YouTube	64	0	0	159.4	0	0	98.6	258	1	3	1.23	1.89

Table 2: Metamorphic relations identified (average of 10 executions). CC: Conjunctive Conditions, DC: Disjunctive Conditions, DP: Disjoint Partitions, CP: Complete Partitions, S: Shuffling, IE: Input Equivalence.

that in some QBS the MRs identified had over 200 follow-up test cases (e.g., due to a `Language` parameter with +300 possible values in IMDb), which is clearly hard to manage and quite inefficient. To avoid this, in our experiments we restricted the maximum number of follow-up test cases to 6.

The experiments were performed on a MacBook Pro laptop equipped with an M1 CPU, 16GB RAM, and 256GB SSD running macOS Big Sur and Java 11. The code and data for replicating the experiment are available on GitHub [2] (release "MET22").

4.2 Experimental results

Table 2 depicts the results of the identification of MRs in the subject QBSs. For each QBS, the table shows the number of source test cases, number of MRs identified from each pattern, number of follow-up test cases, and execution time. As illustrated, the total number of MRs identified ranged from 16.9 in SkyScanner to 1072.9 in IMDb. Not surprisingly, we found that the number of MRs identified increases with the number of query parameters in the QBS and the number of source test cases used as input, except in those cases where there are limit parameters. The average number of follow-up test cases ranged from 1.21 in SkyScanner to 3.55 in Transfermarkt. Execution times remained below 3 seconds in all systems, except IMDb due to the high number of MRs generated (+1K). It is worth highlighting that the applied transformations are based on widely accepted MR patterns for QBSs, and therefore the generated relations should be valid by construction. This means that there should not be false positives, assuming the input specifications are correct. To check the correctness of the generated relations, we manually reviewed 20 MRs for each system (160 in total), among those generated by MRG, finding no inconsistencies.

As an example, the following is one of the MRs automatically generated by MRG on the videogame platform Steam, automatically written using the MR template proposed by Segura et al. [22].

In the domain of Steam (https://store.steampowered.com/search/) **the following metamorphic relation(s) should hold**

- MR77:
  if a source test case is run with the inputs [`sortBy`, `price`, `tag`, `numberOfPlayers`, `feature`]
  and a follow-up test case is constructed by removing the following parameters [`feature`, `tag`]
  and a follow-up test case is constructed by removing the following parameters [`numberOfPlayers`, `price`]
  then the follow-up output(s) should be a (non-strict) superset of the source test output.

5 LIMITATIONS

We identify several limitations to be addressed in future work. First, and more importantly, the fault detection capability of the generated MRs should be empirically evaluated. Their resemblance to the manual MRs used in related papers, however, makes us confidence in their ability to reveal failures. Also, the number of MRs can be very high, which makes it necessary to identify the most relevant MRs, as described in the next section.

In technical terms, MRG only supports assigning a single value to each parameter. In practice, however, we found parameters that can take multiple values at the same time for either filtering the results or expanding them (e.g., parameter `Title Type` in IMDb). Also, two MRs are considered as duplicates when they are identical. However, two MRs could still be very similar if they apply the same input transformations in different order. Hence, implementing more sophisticated methods for removing duplicated MRs will be part of our future work too. Finally, transformations are applied on isolation for simplicity. However, there is potential in combining the different types of transformations (and therefore MR patterns) resulting in more and more complex MRs.

6 OPEN CHALLENGES

Our work opens exciting research challenges, among others:

Application to other domains. One of the strengths of our approach lies on its applicability to potentially any domain where MR patterns can be identified. This could be the case of artificial intelligence applications, for example, where the same types of MRs are frequently reused. Hence, we envision new contributions on the application of similar ideas to other domains enhancing the applicability of metamorphic testing in practice.

Specification inference. The creation of the system specification, and in particular the classification of input parameters—key point of the proposed method—is a manual endeavour. We may remark, however, that this is a one-time effort that clearly pays off once the specification is used for the generation of MRs. Despite this, we envision potential research opportunities on the automated inference and classification of input parameters, e.g., using artificial intelligence methods.

Full test automation. The generation of MRs is the key point for the successful application of metamorphic testing, but not the only one. Automating the whole testing process (MR generation + test case generation + test case execution) would contribute to fully leverage the proposed method and achieve an unprecedented level of automation.

Selection, prioritization, and minimization of MRs. One of the main limitations of our approach lies in the high number of MRs generated. Using all of them is possibly not affordable in practice. Hence, there is a need for selecting those with more chances of revealing failures. Alternatively, they could be reordered (i.e., prioritized) such that failures can be detected as soon as possible, or minimized by identifying redundant MRs that can be safely removed without impacting the fault detection capability of the final test suite [30]. Automatically selecting, prioritizing, and minimizing MRs is therefore an open challenge, especially in those cases where, as in our experiment, the source code is not available. We envision techniques exploiting previous ideas on test case and MR diversity [11, 12, 16].

7 THREATS TO VALIDITY

We identify the following validity threats.

Internal validity. *Are there factors that might affect the results of our evaluation?* For the evaluation, input specifications were manually created from the API specification or the web GUI of the QBSs under test. It is therefore possible that the specifications deviate from the actual QBS's behaviour. To mitigate this threat, each of the specification files were carefully reviewed by at least two authors, and we generated MRs for 8 different real-world systems.

The applied transformations are based on well known MR patterns for QBSs and therefore the generated relations should be valid by construction. It is possible, however, that a bug in our tool would lead to erroneous MRs. To mitigate this threat, we carefully tested our tool (JUnit test cases are available in GitHub [2]). Also, we manually reviewed many of the generated MRs without finding any inconsistency.

External validity. *To what extent can we generalize the findings of our investigation?* We evaluated our approach on a subset of QBSs and therefore our conclusions could not generalize beyond that. To mitigate this threat, we evaluated MRG on 8 popular industrial systems with millions of users worldwide, which makes us confident of the generalizability of our results.

8 RELATED WORK

Some related approaches have also achieved a high degree of automation on the identification of MRs. Kanewala and Bieman [13, 14] proposed several machine learning-based methods for the inference of MRs for scientific programs by analyzing their source code. Zhang et al. [32] presented a search-based algorithm for the inference of polynomial MRs for numerical programs based on the analysis of the program's inputs and outputs. Troya et al. [27] proposed a tool-supported method for the automated generation of MRs for model transformation programs based on the use of patterns and the analysis of execution traces. Ayerdi et al. [6] proposed a multi-objective search algorithm for generating numerical MRs by iteratively modifying test assertions trying to minimize the number of false positives and false negatives with respect to a set of correct and incorrect executions of the program under test. Blasi et al. [8] presented MEMO, a technique and a tool to automatically derive metamorphic equivalence relations from natural language code documentation. Several groups of authors have proposed to create MRs by combining existing ones [18, 29]. Compared to related approaches, our method does not require access to the source code, input/output values, or existing MRs of the program under test. Instead, we rely on a lightweight classification of the input parameters based on existing MR patterns. This allows to efficiently generate MRs without false positives (assuming the input specification is correct), a common limitation of existing approaches.

Liu et al. [19] presented MTKeras, a metamorphic testing tool for machine learning programs built upon the Keras platform and writen in Python. The tool integrates common input transformations (e.g., modifying the input data by addition or multiplication), and output relations (e.g., subsume/subset) based on existing MR patterns. The user can then easily apply MRs over a set of source test cases by combining the supported input transformation and output relations. Their work is closely related to ours in the sense that it exploits existing MR patterns to support the construction of MRs. However, it is still up to the user to define the MRs, whereas in our work those are automatically generated from the specification.

Some approaches aim to assist the tester on the identification of MRs. Sun et al. [26] presented METRIC$^+$, a systematic methodology and a tool to assist testers on the identification of MRs by effectively portioning the input and output domains. A key strength of METRIC$^+$ lies on its scope, being applicable to a wide range of applications. Compared to our work, however, it requires more user involvement for the identification of partitions, constraints, and the definition of the final MRs. Other authors have proposed guidelines for the identification of effective MRs [10, 17] and MR patterns [24, 33] to ease their identification. However, in both cases the identification of the relations is manual, in contrast to our approach, where they are automatically generated.

9 CONCLUSIONS

In this paper, we presented a method and a companion tool for the automated generation of MRs for QBSs starting from a lightweight classification of the input parameters and a set of existing test cases. Input parameters are classified according to their potential to create instances of existing MR patterns. Evaluation results with 8 real-world systems show that hundreds of MRs can be automatically generated in a few seconds. These results open exciting new research directions on the automated generation of MRs driven by the specification and MR patterns.

ACKNOWLEDGMENTS

This work has been supported by the European Commission (FEDER) and Junta de Andalucia under projects EKIPMENT-PLUS (P18-FR-2895), APOLO (US-1264651), and MEMENTO (US-1381595), by the Spanish Government (FEDER/Ministerio de Ciencia e Innovación – Agencia Estatal de Investigación) under project HORATIO (RTI2018-101204-B-C21), and the FPU scholarship program (FPU17/04077).

REFERENCES

[1] [n.d.]. Bikewise: Bicycle Incident Reporting API . https://github.com/bikeindex/bikewise. Accessed Jan 2022.

[2] [n.d.]. MRG (Metamorphic Relation Generator). https://github.com/ssegura/MRG. Accessed Jan 2022.

[3] [n.d.]. OpenAPI Specification. https://www.openapis.org. Accessed Jan 2022.

[4] [n.d.]. Pairwise Independent Combinatorial Testing Tool. https://github.com/microsoft/pict. Accessed Jan 2022.

[5] [n.d.]. RESTful API Modeling Language. http://raml.org/. Accessed Jan 2022.

[6] Jon Ayerdi, Valerio Terragni, Aitor Arrieta, Paolo Tonella, Goiuria Sagardui, and Maite Arratibel. 2021. Generating Metamorphic Relations for Cyber-Physical Systems with Genetic Programming: An Industrial Case Study. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 1264–1274. https://doi.org/10.1145/3468264.3473920

[7] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. 2015. The Oracle Problem in Software Testing: A Survey. *Software Engineering, IEEE Transactions on* 41, 5 (May 2015), 507–525. https://doi.org/10.1109/TSE.2014.2372785

[8] Arianna Blasi, Alessandra Gorla, Michael D. Ernst, Mauro Pezzè, and Antonio Carzaniga. 2021. MeMo: Automatically identifying metamorphic relations in Javadoc comments for test automation. *Journal of Systems and Software* 181 (2021), 111041. https://doi.org/10.1016/j.jss.2021.111041

[9] T. Y. Chen, S. C. Cheung, and S. M. Yiu. 1998. *Metamorphic Testing: A New Approach for Generating Next Test Cases*. Technical Report. Technical Report HKUST-CS98-01, Department of Computer Science, The Hong Kong University of Science and Technology.

[10] T. Y. Chen, D. H. Huang, T. H. Tse, and Z. Q. Zhou. 2004. Case Studies on the Selection of Useful Relations in Metamorphic Testing. In *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC 2004)*. 569–583.

[11] Tsong Yueh Chen, Fei-Ching Kuo, Robert G. Merkel, and T.H. Tse. 2010. Adaptive Random Testing: The ART of test case diversity. *Journal of Systems and Software* 83, 1 (2010), 60–66. https://doi.org/10.1016/j.jss.2009.02.022 SI: Top Scholars.

[12] Hadi Hemmati, Andrea Arcuri, and Lionel Briand. 2013. Achieving Scalable Model-Based Testing through Test Case Diversity. 22, 1 (2013).

[13] U. Kanewala and J. M. Bieman. 2013. Using machine learning techniques to detect metamorphic relations for programs without test oracles. In *IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), 2013*. 1–10. https://doi.org/10.1109/ISSRE.2013.6698899

[14] U. Kanewala, J. M. Bieman, and A. Ben-Hur. 2015. Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels. *Software Testing, Verification and Reliability* (2015). https://doi.org/10.1002/stvr.1594

[15] M. Lindvall, D. Ganesan, R. Ardal, and R.E. Wiegand. 2015. Metamorphic Model-Based Testing Applied on NASA DAT – An Experience Report. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE Int. Conference on*, Vol. 2. 129–138. https://doi.org/10.1109/ICSE.2015.348

[16] Huai Liu, Fei-Ching Kuo, Dave Towey, and Tsong Yueh Chen. 2014. How Effectively Does Metamorphic Testing Alleviate the Oracle Problem? *IEEE Transactions on Software Engineering* 40, 1 (2014), 4–22. https://doi.org/10.1109/TSE.2013.46

[17] H. Liu, F-C. Kuo, D. Towey, and T. Y. Chen. 2014. How Effectively Does Metamorphic Testing Alleviate the Oracle Problem? *Software Engineering, IEEE Transactions on* 40, 1 (Jan 2014), 4–22. https://doi.org/10.1109/TSE.2013.46

[18] H. Liu, X. Liu, and T. Y. Chen. 2012. A New Method for Constructing Metamorphic Relations. In *12th Int. Conference on Quality Software (QSIC), 2012*. 59–68. https://doi.org/10.1109/QSIC.2012.10

[19] Yelin Liu, Yang Liu, Tsong Yueh Chen, and Zhi Quan Zhou. 2020. *A Testing Tool for Machine Learning Applications*. Association for Computing Machinery, New York, NY, USA, 386–387. https://doi.org/10.1145/3387940.3392694

[20] Alberto Martin-Lopez, Sergio Segura, Carlos Muller, and Antonio Ruiz-Cortes. 2021. Specification and Automated Analysis of Inter-Parameter Dependencies in Web APIs. *IEEE Transactions on Services Computing* (2021). In press.

[21] S. Segura. 2018. Metamorphic Testing: Challenges Ahead (Keynote Speech). In *Proceedings of the 3rd International Workshop on Metamorphic Testing (ICSE MET'18)*. ACM, New York, NY, USA, 1. https://doi.org/10.1145/3193977.3193986 Slides: http://personal.us.es/sergiosegura/files/presentations/segura18-MET.pdf.

[22] Sergio Segura, Amador Durán, Javier Troya, and Antonio Ruiz Cortés. 2017. A Template-Based Approach to Describing Metamorphic Relations. In *Proceedings of the 2nd Int. Workshop on Metamorphic Testing (MET '17)*. IEEE Press, 3–9.

[23] Sergio Segura, Amador Durán, Javier Troya, and Antonio Ruiz-Cortés. 2019. Metamorphic Relation Patterns for Query-based Systems. In *Proceedings of the 4th International Workshop on Metamorphic Testing (MET '19)*. IEEE Press, Piscataway, NJ, USA, 24–31. https://doi.org/10.1109/MET.2019.00012

[24] S. Segura, J.A. Parejo, J. Troya, and A. Ruiz-Cortés. 2018. Metamorphic Testing of RESTful Web APIs. *IEEE Transactions on Software Engineering* 44, 11 (Nov 2018), 1083–1099. https://doi.org/10.1109/TSE.2017.2764464

[25] S. Segura, D. Towey, Z. Q. Zhou, and T. Y. Chen. 2018. Metamorphic Testing: Testing the Untestable. *IEEE Software* (2018). https://doi.org/10.1109/MS.2018.2875968

[26] Chang-Ai Sun, An Fu, Pak-Lok Poon, Xiaoyuan Xie, Huai Liu, and Tsong Yueh Chen. 2021. METRIC$^+$: A Metamorphic Relation Identification Technique Based on Input Plus Output Domains. *IEEE Transactions on Software Engineering* 47, 9 (2021), 1764–1785. https://doi.org/10.1109/TSE.2019.2934848

[27] J. Troya, S. Segura, and A. Ruiz-Cortés. 2018. Automated inference of likely metamorphic relations for model transformations. *Journal of Systems and Software* 136 (2018), 188 – 208. https://doi.org/10.1016/j.jss.2017.05.043

[28] E. J. Weyuker. 1982. On Testing Non-Testable Programs. *Comput. J.* 25, 4 (1982), 465–470.

[29] P. Wu. 2005. Iterative Metamorphic Testing. In *29th Annual International Computer Software and Applications Conference, 2005. COMPSAC 2005*, Vol. 1. 19–24.

[30] S. Yoo and M. Harman. 2012. Regression Testing Minimization, Selection and Prioritization: A Survey. 22, 2 (2012). https://doi.org/10.1002/stv.430

[31] Linbin Yu, Yu Lei, Raghu N. Kacker, and D. Richard Kuhn. 2013. ACTS: A Combinatorial Test Generation Tool. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. 370–375.

[32] J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, and H. Mei. 2014. Search-based Inference of Polynomial Metamorphic Relations. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE '14)*. ACM, New York, NY, USA, 701–712. https://doi.org/10.1145/2642937.2642994

[33] Z. Q. Zhou, L. Sun, T. Y. Chen, and D. Towey. 2018. Metamorphic Relations for Enhancing System Understanding and Use. *IEEE Transactions on Software Engineering* (2018). https://doi.org/10.1109/TSE.2018.2876433

[34] Z. Q. Zhou, T. H. Tse, F.-C. Kuo, and T. Y. Chen. 2007. *Automated functional testing of web search engines in the absence of an oracle*. Technical Report TR-2007-06. Department of Computer Science, The University of Hong Kong.

[35] Z. Q. Zhou, S. Xiang, and T. Y. Chen. 2016. Metamorphic Testing for Software Quality Assessment: A Study of Search Engines. *IEEE Transactions on Software Engineering* 42, 3 (March 2016), 264–284. https://doi.org/10.1109/TSE.2015.2478001

[36] Z. Q. Zhou, S. Zhang, M. Hagenbuchner, T. H. Tse, F-C. Kuo, and T. Y. Chen. 2012. Automated Functional Testing of Online Search Services. *Software Testing, Verification and Reliability* 22, 4 (June 2012), 221–243. https://doi.org/10.1002/stvr.437