An experimental analysis of a near-optimal graph-based test generation algorithm

Mutlu Beyazıt[1], Serge Demeyer[1]

[1]Universiteit Antwerpen and Flanders Make vzw, Belgium

Abstract

Graphs are widely used in model-based test generation. A commonly used family of coverage criteria is based on covering edge sequences of length k for $k \geq 1$. A near-optimal test generation algorithm (which based on solving Chinese postman problem) has a known algorithmic complexity of $\mathcal{O}(|V|^{3k})$. While we cannot change the inherent complexity of the algorithm, heuristics can heavily improve upon the actual runtime of the algorithm. In this paper, for the purpose of developing such a heuristic, we illustrate the potential points of improvement by means of an experimental analysis. Our results suggest that the good coverage properties displayed by the algorithm are prone to being exploited to improve its runtime performance at least for certain types of graphs.

Keywords

model-based testing, graph model, test generation, coverage, experimental analysis

1. Introduction

Model-based testing is the activity of testing a (software) system based on the abstractions called models or specifications. It is commonly used for functional testing of the system [1], and it involves activites such as modeling, test generation, test concretization and test execution [2]. There have been many studies reporting the benefits of model-based testing such as increased effectiveness and efficiency in terms of fault detection and testing costs [3, 4]. Especially relevant for fast evolving systems, (model-based) test generation has been used in the context of Graphical User Interfaces [5, 6], web services [7], web applications [2], mobile applications [8], etc.

There are different types of graph-based models, which differ from each other in terms of expressive power and the elements on which they put the focus. Two of these types are *state-based models*, such as finite state machines [9, 10], and *event-based models*, such as event flow graphs [11] and event sequence graphs [12].

Test generation is an important part of model-based testing. For graph-based models, the idea is in general to extract edge sequences. Extracted sequences are supposed to satisfy some coverage criteria based on, for example, coverage of certain faults or coverage of edge sequences of different lengths [13]. Although it is more costly, generation of small test sets is desirable in the context of fast evolving applications because the (regression) tests are executed against every minor release.

BENEVOL 2023 The 22nd Belgium-Netherlands Software Evolution Workshop Nijmegen, 27-28 November 2023
mutlu.beyazit@uantwerpen.be (M. Beyazıt); serge.demeyer@uantwerpen.be (S. Demeyer)

This paper studies a commonly used near-optimal graph-based test generation algorithm [14, 15, 6, 8]. This algorithm generates a set of test cases covering edge sequences of length $k \geq 1$ by solving Chinese postman problem [16]. The algorithm has a known complexity of $\mathcal{O}(|V|^{3k})$ [14, 17] where $|V|$ is the number of vertices in the graph. Thus, its runtime increases very fast especially when the parameter k is increased. However, one can use heuristics to improve the actual runtime of the algorithm. The aim of the paper is to motivate the exploration of such a heuristic and experimentally validate the possibility of reducing the actual runtime. The heuristic considered in this paper is based on utilizing test cases achieving weaker coverage in generation of test cases achieving stronger coverage. We investigate whether tests generated using the algorithm show good coverage properties for also the coverage targets which are out of their scope. Furthermore, since the exploration of heuristics is also dependent on which part of the test generation algorithm is the most time-consuming part, we perform an experimental runtime analysis of the algorithm.

2. Background

This section gives a brief background on graphs, graph-based coverage criteria and the graph-based test generation algorithm of interest.

2.1. Graphs

We start by giving the definition of a graph.

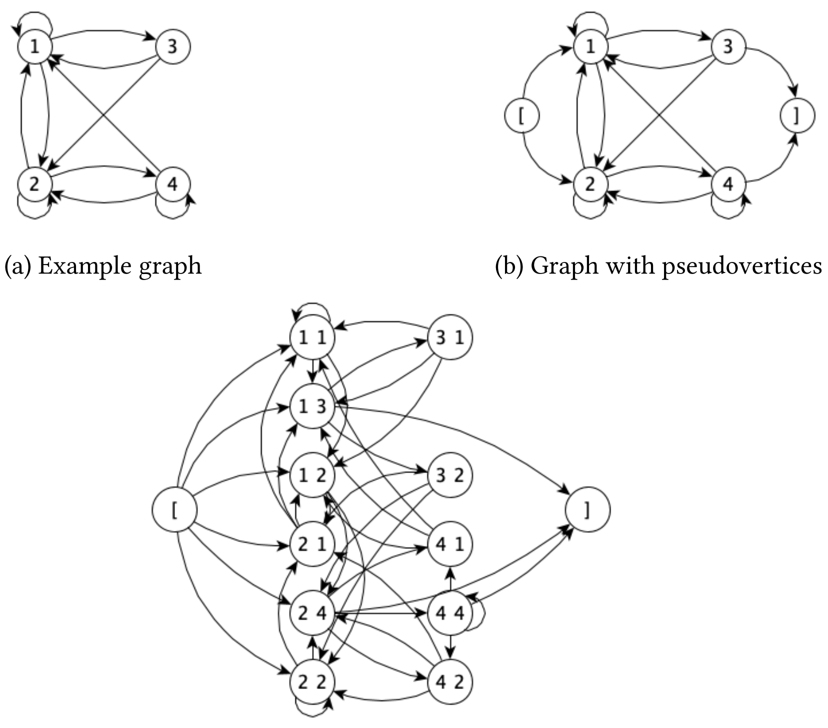
Definition 1. *A directed multigraph is a tuple $G = (V, E)$ where $V \neq \emptyset$ is a finite set of vertices, and E is a finite multiset of edges where each edge is an order pair in $V \times V$. A walk $v_1, v_2, \ldots, v_l$ in G is a sequence of vertices where each $(v_i, v_{i+1}) \in E$.*

Depending on the context, it is possible to enrichen the semantics of a graph-based model by distinguishing between different types of vertices and/or by labeling the vertices or edges with different elements. For example, in state-based models, vertices represent states and edges represent transitions where each transition is also labeled by inputs or input-output pairs. In event-based models, on the other hand, vertices represents events, and edges have no labels; they define a ”follows” relation on events. Of course, it is always possible to define extensions by introducing guards, quiescence, time, hiearchy, etc. Figure 1a shows an example graph.

2.2. k-Edge Coverage

Regardless of the type of the graph-based model, a common purpose is to extract or generate subgraphs from the given graph to be used as test cases. The way the test cases are generated is dependent on the model elements. For example, a model with guards is harder to work with than one without guards because it is more expressive. Still, in many practical applications, the generated subgraphs are linear and, thus, walks in the graph are used as test cases.

In the generation of test cases, coverage criteria of different strengths are defined and used. One way to do this is to utilize k-edges. A k-edge is a walk of length k where $k \geq 1$, and k-edges are used as coverage targets. The associated coverage criterion is given in Definition 2.



(a) Example graph

(b) Graph with pseudovertices

(c) One-time transformed graph

Figure 1: Example graphs

Definition 2. *Given a graph G and a set of test cases T, T is said to cover a k-edge s in G if s appears in one of the the test cases in T. T is said to achieve k-edge coverage if it covers all k-edges in G.*

Note that, if it is possible to extend each walk in a given graph to a longer walk, the set of k-edges is always a subset of the set of $(k + 1)$-edges for each $k \geq 1$. Consequently, $(k + 1)$-edge coverage is stronger than k-edge coverage. Thus, parameter k is used to assess the strength of the set of test cases T.

2.3. Test Generation Covering k-Edges

In certain contexts or, generally, for generation of test cases, certain vertices in the graph can be marked as inital/start/entry or final/finish/exit vertices. Furthermore, the graph model may be required to satisfy certain properties such as consistency where each vertex is reachable from an entry vertex and an exit vertex is reachable from each vertex.

The test generation algorithm we consider in this work has the following assumptions.

- The graph has at least one entry vertex and one exit vertex.
- The graph is consistent.
- Each test case is a walk from an entry vertex to an exit vertex.

Without any loss of generality, we use one pseudostart vertex denoted by ”[” to mark the entry vertices and one pseudofinish vertex denoted by ”]” to mark the exit vertices. Each entry vertex has an incoming edge from the pseudostart vertex and each exit vertex has an outgoing edge to the pseudofinish vertex. Figure 1b shows inclusion of pseudo vertices to the graph in Figure 1a assuming that ”1” and ”2” are entry vertices, and ”3” and ”4” are exit vertices.

The main steps of the test generation algorithm is given in Algorithm 1 in two parts. In ”graph transformation” part of the algorithm (See Part 1 in Algorithm 1), G is transformed $k-1$ times to obtain a graph G_t such that, when the edges of G_t are covered, k-edges of G are covered. In other words, each edge in G_t corresponds to k consecutive edges in G. It has the time complexity of $\mathcal{O}(k^2|V|^{2k})$; details can be found in [15, 17]. Figure 1c shows a graph which is obtained by transforming the graph in Figure 1b once.

Algorithm 1 Algorithm to Generate Test Cases Achieving k-Edge Coverage

```
Require: A consistent graph $G = (V, E)$ and coverage parameter $k \geq 1$
Ensure: Generated set of test cases $T$ achieves $k$-edge coverage
 1: /* Part 1: Graph transformation */
 2: $G_t \leftarrow G$
 3: $i \leftarrow 1$
 4: while $i < k$ do
 5:     $G_t \leftarrow$ Transform $G_t$ using $G$
 6:     $i \leftarrow i + 1$
 7: end while
 8: /* Part 2: Euler path generation */
 9: $G_{sc} \leftarrow$ Add the edge $(], [)$ to $G_t$
10: $G_b \leftarrow$ Add proper paths until degree of each vertex is 0
11: $T \leftarrow$ Compute Euler paths from [ to ] in $G_b$
```

Once the transformed graph is computed, ”Euler path generation” part of the algorithm (see Part 2 in Algorithm 1) works on solving the Chinese postman problem [16, 18]. First, a backward edge from the pseudofinish vertex to the pseudostart vertex is added to have a strongly-connected graph (Line 7). Then, the graph is balanced by repeating some of the existing paths (Line 8). This is done by matching each vertex with positive degree with a vertex with negative degree by solving an assignment problem [19, 20] and inserting a shortest path from a vertex with positive degree to its matching vertex with negative degree. Finally, after the graph is balanced, paths from the pseudostart vertex to the pseudofinish vertex are computed by visiting each edge exactly once (Line 9). These paths are called Euler paths and they correspond to the test cases. Line 10 detemines the complexity of this part, and it is given by $\mathcal{O}(|V|^{3k})$ [14]. Compared to the complexity of Part 1, Part 2 is likely the most time consuming part because, in practice, $k^2 << |V|$ and, also, hidden constants that are neglected in big O notation have more effect on the runtime of Part 2.

Note that Algorithm 1 may yield infeasible paths depending on the semantics of the actual graph model. In this paper, we assume that all paths are feasible.

3. Experimental Analysis

In this section, we perform experiments to collect data on the test generation algorithm given in 1 using randomly generated graph models. Our aim is to identify possible opportunities to improve the near-optimal algorithm in terms of runtime time.

3.1. Research Questions

In order to identify possible points of improvements for Algorithm 1, we formulate the following research questions.

RQ1 Can we confirm that Euler path generation part of Algorithm 1 is the most time consuming part?

RQ2 Given a test set achieving k-edge coverage generated by Algorithm 1, what factors affect the coverage of m-edges and how well does the test set cover m-edges where $m > k$?

RQ2 is formulated to see whether test cases achieving k-edge coverage ($k \geq 1$) show good coverage properties for also the coverage targets which are out of their scope. This is an indication of possibility of utilizing k-edge covering test cases in generation of the test cases achieving m-edge coverage for $m > k$. However, this possibility is also dependent on the change in the runtime of different parts of Algorithm 1 when k is increased. Thus, we also define RQ1 to investigate which of the two main parts of the algorithm takes more time: graph transformation part or Euler path generation part.

3.2. Experimental Setup

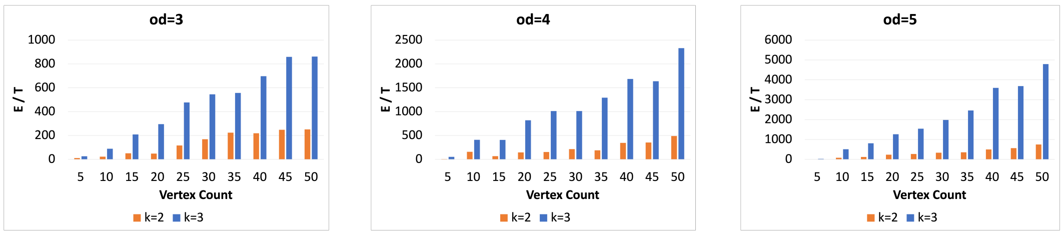
In order to collect data, we used randomly generated and consistent graph models satisfying the following conditions.

- Each graph has one entry vertex and one exit vertex. This is done to reduce the possible effects of multiple pseudoedges on the the test generation. Although there is no detailed study on the subject, Algorithm 1 may yield non-optimal results if the number of pseudoedges is greater than 1.
- All vertices in a graph have the same number of outgoing edges. This is done to control the model size properly, to increase the probability of generating a consistent graph, and to ensure that each edge takes part in $k-$edges in a more uniform manner.

We use the following parameter values in our experiments.

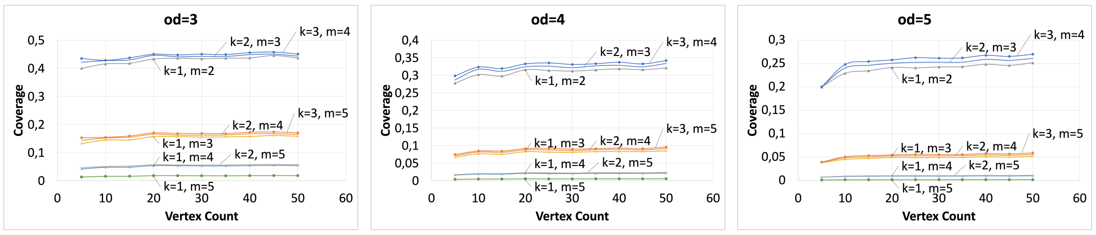
- Number of vertices in a graph, or simply vertex count: $vc = 5, 10, 15, 20, 25, 30, 35, 40, 45, 50$
- Number of outgoing edges of a vertex, or simply out degree: $od = 3, 4, 5$
- k-edge coverage parameter: $k = 1, 2, 3$
- Parameter for coverage targets: $m = 1, 2, 3, 4, 5$

For each (vc, od, k) triple, we generate 10 random graphs and perform 3 test generation runs to collect data in a feasible time period. All runs are executed on a MacBook Pro 14 with Apple M1 Pro CPU, 16GB RAM and macOS Monterey operating system.



(a) E/T values for $od = 3$ (b) E/T values for $od = 4$ (c) E/T values for $od = 5$

Figure 2: E/T values



(a) $cov(k, m, od)$ values for $od = 3$ (b) $cov(k, m, od)$ values for $od = 4$ (c) $cov(k, m, od)$ values for $od = 5$

Figure 3: $cov(k, m, od)$ values

3.3. Results

Let E denote the time it takes to generate the Eulerian paths and and T denote the time to perform the graph transformation part of Algorithm 1. To see how E and T values compare with respect to each other, we give bar plots which include E/T values for $k = 2, 3$ and $od = 3, 4, 5$ in Figures 2a, 2b and 2c. Note that values for $k = 1$ are not included since no transformation takes place to cover 1-edges.

RQ1 can be answered by using these plots. It is clear that Eulerian path generation part of Algorithm 1 is far more time consuming than its graph transformation part. The difference increases for increasing k values and increasing od values.

The results of coverage analysis of k-edge covering test sets for $k = 1, 2, 3$ are given in Figures 3a, 3b and 3c for $m = k + 1, \dots, 5$. While plotting the graphs, average of the values obtained over 10 random graphs with 3 runs for each graph are used.

To answer RQ2, let $cov(k, m, od)$ denote the ratio of m-edges covered by the test set achieving k-edge coverage for a given value of od. Figures 3a, 3b and 3c show that, when od and $m - k$ are fixed, $cov(k, m, od)$ values obtained using different k are very similar and they are not dependent on vc. Also, promisingly high $cov(k, m, od)$ values are observed especially when $m = k + 1$ and od is small: $cov(k, k + 1, 3) = 43\%$, $cov(k, k + 1, 4) = 32\%$ and $cov(k, k + 1, 5) = 25\%$. Thus, there is a decreasing trend for $cov(k, m, od)$ when od is increased; however, more data is required to make further comments on the nature of this trend. On the other hand, $cov(k, m, od)$ shows a faster-than-linear decrease for increasing m for different values of od when k is fixed.

4. Concluding Remarks

Our results have both positive and negative implications for improvement of the near optimal graph-based test generation algorithm given in Algorithm 1.
On the positive side, we have the following.

- A test set achieving k-edge coverage covers a significant portion of $k + 1$-edges already. Therefore, it might be possible to develop heuristics utilizing test set achieving k-edge coverage in the generation of test sets achieving $k + 1$-edge coverage. This my cause a considerable speed-up in test generation, because a test achieving k-edge coverage can be generated much faster than a tet set achieving $k + 1$-edge coverage, especially when k increases.
- Part 2 (Eulerian path generation) of the algorithm consumes significantly more time than Part 1 (graph transformation). This is a positive result, because if Part 1 is a combinatorial algorithm which is based on generating a graph containing coverage targets. Thus, it is not likely to make use of the good coverage properties displayed by the test sets to improve this part.

On the negative side, the results suggest that the portion of $k + 1$-edges covered by test sets achieving k-edge coverage may decrease fast when the average outgoing degree of the graph-based model increases. Thus, heuristics developed based on the properties discussed above may not work under certain conditions.

References

[1] B. Beizer, Software Testing Techniques (2nd Ed.), Van Nostrand Reinhold Co., USA, 1990.

[2] G. R. Mattiello, A. T. Endo, Model-based testing leveraged for automated web tests, Software Quality Journal 30 (2022) 621–649. URL: https://doi.org/10.1007/s11219-021-09575-w. doi:10.1007/s11219-021-09575-w.

[3] M. Utting, B. Legeard, Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[4] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A. J. H. Simons, S. Vilkomir, M. R. Woodward, H. Zedan, Using formal specifications to support testing, ACM Comput. Surv. 41 (2009). URL: https://doi.org/10.1145/1459352.1459354. doi:10.1145/1459352.1459354.

[5] A. M. Memon, Q. Xie, Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software, IEEE Trans. Softw. Eng. 31 (2005) 884–896. URL: https://doi.org/10.1109/TSE.2005.117. doi:10.1109/TSE.2005.117.

[6] F. Belli, M. Beyazit, N. Güler, Event-oriented, model-based GUI testing and reliability assessment—approach and case study, volume 85 of *Advances in Computers*, Elsevier, 2012, pp. 277–326. URL: https://www.sciencedirect.com/science/article/pii/B9780123965264000060. doi:https://doi.org/10.1016/B978-0-12-396526-4.00006-0.

[7] L. Frantzen, J. Tretmans, Towards model-based testing of web services, 2006. URL: https://api.semanticscholar.org/CorpusID:16371114.

[8] G. de Cleva Farto, A. T. Endo, Evaluating the model-based testing approach in the context of mobile applications, Electronic Notes in Theoretical Computer Science 314 (2015) 3–21. URL: https://www.sciencedirect.com/science/article/pii/S1571066115000250. doi:https://doi.org/10.1016/j.entcs.2015.05.002, cLEI 2014, the XL Latin American Conference in Informatic.

[9] G. H. Mealy, A method for synthesizing sequential circuits, The Bell System Technical Journal 34 (1955) 1045–1079. doi:10.1002/j.1538-7305.1955.tb03788.x.

[10] E. F. Moore, et al., Gedanken-experiments on sequential machines, Automata studies 34 (1956) 129–153.

[11] Q. Xie, A. M. Memon, Using a pilot study to derive a GUI model for automated testing, ACM Trans. Softw. Eng. Methodol. 18 (2008). URL: https://doi.org/10.1145/1416563.1416567. doi:10.1145/1416563.1416567.

[12] F. Belli, C. J. Budnik, Minimal spanning set for coverage testing of interactive systems, in: Proceedings of the First International Conference on Theoretical Aspects of Computing, ICTAC'04, Springer-Verlag, Berlin, Heidelberg, 2004, pp. 220–234. URL: https://doi.org/10.1007/978-3-540-31862-0_17. doi:10.1007/978-3-540-31862-0_17.

[13] F. Belli, M. Beyazıt, A. T. Endo, A. Mathur, A. Simao, Fault domain-based testing in imperfect situations: A heuristic approach and case studies, Software Quality Journal 23 (2015) 423–452. URL: https://doi.org/10.1007/s11219-014-9242-6. doi:10.1007/s11219-014-9242-6.

[14] F. Belli, C. J. Budnik, Test minimization for human-computer interaction, Applied Intelligence 26 (2007) 161–174. URL: https://doi.org/10.1007/s10489-006-0008-0. doi:10.1007/s10489-006-0008-0.

[15] F. Belli, A. Hollmann, Test generation and minimization with "basic" statecharts, in: Proceedings of the 2008 ACM Symposium on Applied Computing, SAC '08, Association for Computing Machinery, New York, NY, USA, 2008, pp. 718–723. URL: https://doi.org/10.1145/1363686.1363856. doi:10.1145/1363686.1363856.

[16] A. Aho, A. Dahbura, D. Lee, M. Uyar, An optimization technique for protocol conformance test generation based on UIO sequences and rural chinese postman tours, IEEE Transactions on Communications 39 (1991) 1604–1615. doi:10.1109/26.111442.

[17] F. Belli, M. Beyazıt, Exploiting model morphology for event-based testing, IEEE Transactions on Software Engineering 41 (2015) 113–134. doi:10.1109/TSE.2014.2360690.

[18] H. Thimbleby, The directed chinese postman problem, Software: Practice and Experience 33 (2003) 1081–1096. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.540. doi:https://doi.org/10.1002/spe.540. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.540.

[19] H. W. Kuhn, The hungarian method for the assignment problem, Naval Research Logistics Quarterly 2 (1955) 83–97. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109. doi:https://doi.org/10.1002/nav.3800020109. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800020109.

[20] J. Munkres, Algorithms for the assignment and transportation problems, Journal of the Society for Industrial and Applied Mathematics 5 (1957) 32–38. URL: http://www.jstor.org/stable/2098689.