A structural approach to assess graph-based exercises

José Paulo Leal and Rúben Sousa

CRACS & INESC-Porto LA, Faculty of Sciences,
University of Porto
Porto, Portugal
zp@dcc.fc.up.pt up201001961@fc.up.pt

Abstract. This paper proposes a structure driven approach to assess graph-based exercises. Given two graphs, a solution and an attempt of a student, this approach computes a mapping between the node sets of both graphs that maximizes the student's grade, as well as a description of the differences between them. The proposed algorithm uses heuristics to test the most promising mappings first and prune the remaining when it is sure that a better mapping cannot be computed.
This algorithm is applicable to any type of document that can parsed into its graph-inspired data model. This data model is able to accommodate diagram languages, such as UML or ER diagrams, for which this kind of assessment is typically used. However, the motivation for developing this algorithm is to combine it with other assessment models, such as test case model used for programming language assessment.
The proposed algorithm was validated with thousands of graphs with different features produced by a synthetic data generator. Several experiments were designed to analyse the impact of different features such as graph size, and amount of difference between solution and attempt.

Keywords: automatic assessment, graph comparison, graph-based exercises

1 Introduction

Graphs are mathematical structures that model relationships among objects. They can be used in a wide range of domains such as network topology, software architecture, digital circuit design, just to mention a few. Diagrams are an apt example of a document type with a graph-based representation that requires automatic assessment. However, graphs can be used for assessing exercises where the relationship among parts is important but not determinant. Finite Deterministic Automata (FDA) and even programming languages are examples of this kind of assessment.

The assessment of an FDA should be twofold [2], based on the recognized strings and on the structure of the state automaton. If an FDA recognizes all strings it should, and only those, then it must be correct. Otherwise, examples of the strings that should be recognized but are not, and vice versa, can be automatically generated. However, these examples seldom contribute to overcome the error. An helpful feedback must pinpoint what is wrong. For instance, what nodes are missing or what transitions must be removed. This can be achieved using graph assessment since the structure of an FDA can be represented by a state automaton [2].

Programming language assessment would also benefit from a similar approach. The standard way of assessing a program [4] is to compile and then execute it with a set of test cases. A program is considered correct if it compiles without errors and the output of each execution is what is expected. If it is incorrect then the most this approach can provide are examples of input that generates wrong output. It cannot pinpoint the errors in the code of the program. An attempt to make this kind of assessment should be based on the structure of the program, specifically its abstract semantic graph.

The ultimate goal of the research presented in this paper is to define a general methodology for assessing graph-based exercises, applicable in a wide range of domains including FDAs, programming languages but also diagrams. The objective of this paper is to propose a graph assessment algorithm and to evaluate its efficiency for graphs with the size typically used in automated assessment.

The proposed assessment algorithm is based on the *graph structure*, meaning that it actually computes the mapping that achieves the best grade. To avoid checking a large number of mappings it iterates over them considering first the most promising. The mappings are iterated in an order that allows the algorithm to prune the majority of them, when it can ensure that the remaining mappings cannot produce a grade higher than the best one computed so far. The iteration order is driven by the types and properties of nodes.

The remainder of this paper is organized as follows. Section 2 surveys the existing literature on assessment of graph-based exercises. Section 3 describes the proposed algorithm, including the definition of the data structures to represent graph based exercises and their assessment. This approach is validated in Section 4 using a graph generator to test the limits of the applicability of the proposed methodology. Finally, Section 5 highlights both the main contributions of this paper and the work ahead to apply this form of assessment in different scenarios.

2 Related work

This section surveys the existing literature on automatic diagram assessment. To the best of the authors' knowledge, no general algorithm for assessment of graph-based documents has yet been proposed. The existing proposals are targeted solely to diagrams, and focus mostly their labels rather than their structure.

Most of the available automatic diagram assessment systems were designed for a specific diagram type, Examples of these single types addressed by existing systems are deterministic finite automata (DFA) [2, 5], UML class diagrams [1, 6], UML use case diagrams [9], Entity-Relationship diagrams [3], among others.

All these systems determine a mapping between nodes of solution graph and nodes the student's answer. The easiest approach is to use a fix set of labels in both graphs. For instance, the exercise descriptions used in assessment system proposed by Soler [6] for UML class diagrams must fix the class names used by students. Finding a mapping between the node sets of both graph is thus straightforward. A variant of this approach is the assessment in stages that Ali et al. [1] proposed. This system will not advance to the next stage until the current one is completely correct, otherwise it reports feedback on what is wrong or missing. The considered stages are: structural analysis, verification process and a language checking. The first stage compares the number of nodes, attributes and operation and their types. The second stage checks if connections have the correct type source and target. Knowing that the graph structure is correct (by the two stages above), the system checks if the labels in nodes and attributes are nouns and in the operations are verbs.

The automata-base graph analyser of Shukur and Mohamed [5] works in a way that is similar to that presented above. The system does two types of evaluation: a static one and a dynamic one. The static analysis is made by comparing the global number of states, the number of initial and final states and the number of connections. The dynamic analysis is made by testing two sets of strings. One of the sets is composed by strings that the model should accept. If any of it is rejected, the graph is not correct. The second set is composed by strings that should be rejected by the system. So, by opposition, if any string is accepted the graph is not correct.

The work of Thomas, Smith and Waugh [8, 7] describes a system with similarities with the approach presented in this paper. It is a generic system able to handle different diagram types. Elements can be represented as boxes or circles and each connections as lines. The system tries to match those elements from students' answer to the elements of the solution. For each pair of nodes and edges is computed a similarity measure and with that value the system can assume what is (or not) a valid match. If the similarity is high, the system assumes it as correct. On the other hand, if the similarity is low the system is marked as not correct. The approach that will be presented in here describes a way how to find the best match without these assumptions.

3 Graph assessment algorithm

The objective of the algorithm described in this section is to assess an exercise represented as an extended graph by comparing it with a standard solution, represented also as an extended graph. The assessment consists in determining a set of differences between both extended graphs. These differences can be summarized in a grade, a numerical value within a fixed range (e.g. 0 to 100). If the set of differences is empty then the attempt of the student reaches the maximum grade; otherwise each difference introduces a penalty according to its type. For instance, a missing node may have a higher impact on the grade than a missing edge. Wrong types, missing or wrong properties have also their own penalties, depending on the language.

The basic approach is to determine the mapping between the node sets of both extended graphs that maximizes the grade. This can be solved by a simple generate and test strategy. If one generates all the possible mappings between both extended graphs, for each mapping one can determine the differences between graphs and synthesize a grade. After iterating through all possible mapping one can select the one that produces the highest grade.

The problem with this approach is the number of mappings. If both extended graphs have n nodes then the number of mappings is $n!$, rending this approach unfeasible for graphs with more than 12 nodes. The proposed solution is to generate mappings if a particular order, with a decreasing contribution of the nodes to the final grade. The heuristic is that a node mapping with a high contribution to the grade has higher chances of being selected for the best mapping. Of course that this heuristic is only helpful if it is combined with the pruning of a significant number of mappings. That is, if alter testing a reasonably small number of mappings one can discard most of the remaining mappings.

A solution and and attempt with equal sizes, i.e. with an equal number of nodes, is a particular case. In general these graphs have different sizes since the student may have omitted nodes or edges, or introduced unneeded ones. In this case the approach is to reduce the number of nodes in one graph until both have the same size; edges connecting the removed nodes are also removed. If one graph has n nodes and the other has m modes, with $n > m$, then there are $n!/m!$ different ways to make them equal. Again, the approach is to delete first the nodes that are least expected in the mapping, and pruning the tail of the node deletion list once it is certain that those alternatives cannot contribute to improve the best mapping determined so far.

This section details several parts of the proposed algorithm. Subsection 3.1 introduces the definitions of extended graph and graphs differences, and defines the computation of a grade from a set of graph differences. Subsection 3.2 explains how two generate node mappings with a decreasing node contribution to the grade.

3.1 Data structures

The proposed algorithm processes two *extended graphs*, a standard solution and a student attempt. A simple graph $G = (N, E)$ is defined by a set of nodes and a set of edges, where an edge is a pair of nodes. In an extended graph both nodes and edges have a *type* and a set of *properties*. An extended graph is a multigraph, in the sense that a pair of nodes may have more than a one edge, possible with different types.

Node and edge types capture the essential features of a graph-based language. Take UML diagrams for instance. Each kind of diagram combines nodes and edges of particular kinds. A use case diagram has as node types actor and use case, and as edge types associations, dependencies and generalizations. The features that are not captured by types are encoded as properties. Properties are simply name value-pairs. Consider an association in an UML class diagram; it may have a navigability, multiplicities, roles and other kinds of properties.

The assessment of an extended graph against another is a *set of differences*. The most relevant differences are detected when both graphs are made of the same size, such as insertion and deletion of nodes. The rest of the differences are computed based on a mapping between the nodes set of both extended graphs. Consider a mapping $m : N \to N'$ and the nodes $a \in N$ from the extended graph used as solution. If a and $m(a)$ are indistinguishable then not difference is added to the set. Otherwise, a differences of a certain kind is signaled: if the types of a and $m(a)$ differ, $m(a)$ has a wrong type; if a property of a and $m(a)$ differs, a property insertion, deletion or wrong value is signaled.

The assessment restricted to nodes plays an important role in the proposed algorithm, since it is quicker to compute. However, the a complete assessment must also consider edges. When nodes are removed to make force graph to have the same size, the insertion and deletion of arcs connecting then is also detected. The rest of the arcs depends on the mapping. For each $(a, b) \in E$ is expected an $(m(a), m(b)) \in E'$ and vice versa. Otherwise edge insertions, omissions, wrong type, as well as edge property differences, are also signaled.

Given a set of differences it is possible to compute a grade. The empty difference set has the maximum grade (e.g. 100). Each kind of difference has a certain penalty and these a grade is computed by subtracting these penalties to the maximum grade. Penalties depend on their kind and the size of the graph. In general a difference in a node should have a higher impact that a difference on a edge, but ultimately this depends on the graph language. There are a number of weights that have to be tuned for a particular language, using actual grades given by experts as benchmark. The same penalty has different impacts according to the solution graph size. For instance, a missing node will have higher impact on a small graph than on a large graph.

Grades computed from set of differences are much more than just the final output of the assessment algorithm. They are essential to control it, in particular the grade restricted to the node contribution, as is explained in the next subsection.

3.2 Node mappings

The general strategy of assessing an extended graph against another is to determine a mapping between then that produces the higher grade. Due to the large number of possible mappings it is important to have heuristics to consider the most promising first and to have a criteria to prune most of them all.

The node component of the assessment outweighs the edge component, although its computational complexity is much smaller. In a graph with n nodes, there are n^2 pairs or nodes, although these can be combined in $n!$ mappings. If one iterates over the set or mappings by decreasing order of their node contributions to the grade (i.e. with less penalties), then the first mappings have higher chance of being the best than those appearing afterwards.

The first step for generating these mappings is to compute the contribution for the grade of individual node mappings. The initial mapping candidate is constructed from the individual mappings with best grade (less differences) for each node in the standard node set. Note that this is a mapping candidate, it may not be a valid candidate if two different nodes are mapped in the same node. The rest of the individual mappings are sorted by decreasing order of the difference of their contributions

4 Validation

The graph assessment algorithm described in the previous section was implemented in Java 1.8. This implementation was used in a number of experiments to validate the applicability of the proposed approach in the assessment exercises on graph-based languages.

The validation was conducted using synthetic graphs. This approach is different from the validations described in the existing literature on diagram assessment systems, surveyed in section 2. Most of the referred authors use actual exercises and student attempts, or a corpus with a large number of diagrams.

The reason for choosing synthetic data to validate this approach is twofold. Firstly, it is not intended for a specific graph-based language and should be adjustable to any graph-based languages that fits in the extended graph data model. Hence, its is important to test it with a wide range of settings, varying the number of types and properties, as will happen with different graph-based languages. Secondly, it is important to test the limits of the proposed approach, in terms of graph sizes and amount of difference between and attempt graphs, for which a large number of graphs pairs is required.

4.1 Graph generator

The graph generator is a component that produces synthetic graphs for testing and validating the proposed graph assessment algorithm. This component is used to generate both a solution graph and attempts near to a given solution. The attempt graph cannot be another random graph, it must be close enough to the solution to be able to produce a meaningful correction.

The graph generator follows the builder design pattern. It has a number of setting that control of the minimum and maximum number of nodes, types and properties. The number of edges for a graph with n nodes are $n-1$ and $n(n+1)/2$ since these are the minimum and maximum number of edges for a connected graph with n nodes. When a new graph is requested, its nodes and arcs with respective types and properties are randomly generated within these boundaries.

Graphs used in graph-based languages are typically connected graphs. Thus, the generator ensures that generated graphs are connected. It computes the connected components of the graph and, while it has more then one, it replaces a *redundant* edge in one component with an edge to a node in a different component. A redundant edge in a connected component is one that can be removed without breaking connectivity.

As explained above, the graph generator can also be used to produce graphs within the vicinity of a given graph, i.e. with a given number of variations, in number of nodes, edges, types and parameters. Within these boundaries the generator: inserts or removes nodes; changes types; inserts, removes or changes properties. Since the graphs produced this way are modelling student attempts, they may be disconnected graphs.

When producing a graph variant to model a student attempt, the generator produces also a set of differences. This set of differences uses the same type of data structure returned by the assessment method. Hence it is straightforward to compare the differences detected by the assessment method with those produced by the generator. This comparison validates the algorithm and its implementation.

Not all student attempts are wrong. Some may be equivalent to the standard solution, and this situation must also be tested. Nevertheless, it would be highly improbable for the two graphs to have nodes and arcs exactly in the same order. Comparing two exactly equal graphs could have an influence of the performance of the algorithm. Thus, the nodes and arcs of variant graphs are always shuffled, independently of the difference between the original and the variant graph.

4.2 Experiments

The implementation of the proposed graph assessment algorithm and synthetic graph generator described in the previous subsection were used in a series of experiments that are reported in this sub-section. These experiments were designed to answer the following questions.

Up to what graph size can this algorithm be used? The complexity of the graph homeomorphism problem is unknown, but most likely it is high enough to prevent the use of this approach on graphs above a certain size.

Do heuristics have a significant impact on performance? The heuristics were designed to explore the most promising mappings first, but they have an initial cost and depend of the effectiveness of the pruning criteria.

What is the impact of weights in performance? The algorithm is driven by grades and heuristics reply on the contribution of nodes to grades. The balance between the weight of node and edge grades is bound to influence performance.

What is the impact of domain specific data? The algorithm was designed to take advantage of the types and properties assigned to nodes and edges. This data makes node and edge easier to identify and the algorithm should perform better as more of it is available.

How dissimilar can solutions and attempts be? If the attempt and the solution are completely dissimilar then it makes no sense to compute differences between then and the grade should be zero. However, the assessment algorithm should perform well for attempts within a certain range of the solution.

The experiments that answered these questions ran on a computer with 4 cores and 8 i7-3630QM CPU at 2.40GHz, with 8 GByte of RAM . For each setting the experiment was repeated with 100 different random pairs of graphs. On most cases the assessment of a pair of graphs was executed well bellow 1/2 a second. Occasionally some pairs of graphs take a longer time hence a timeout of 2 seconds was. In these case the assessment was considered *incomplete*, although the result obtained within the allotted time may actually be correct.

The first experiment addressed the size of graphs that can be assessed with the proposed algorithm. Alur et al. argue that graphs in used exercises are usually small [2] and thus this complexity is not a serious problem. This appears to be the case in DFA, the domain they studied, and also many other domains, such as UML class and use case diagrams. However, an Entity-Relationship exercise to model a simple database may have have more than 20 nodes. The results obtained with hundreds of equivalent graphs pairs show that the proposed algorithms deals with orders of up to 30.

Another addressed the impact of pruning. For that purpose a variant of mappings iterator was implemented. This iterator returns all the possibles mappings, without the initial overhead required by the iterator of the proposed algorithm. The rest of the algorithm was maintained unchanged. This algorithm was tested with equivalent graphs but could only complete the assessment of graphs of grade 6 or lower. Pairs of graphs with a larger number of nodes produce always an incomplete assessment. This compares with the use of the optimized iterator that can assess graph pairs up to order 30.

The third experiment addressed the impact of weights, in particular the balance between the node and edge contribution to assessment. Since the heuristics rely on the node contributions to prune, a larger contribution of edges decreases efficiency. It should be noted that the actual weights will depend of the specific graph-based language and on particular grading criteria defined by the teacher. In any event, it is expected that nodes contribute at least with half of the grade and in general with less than that. In fact, an equal weight of nodes and edges produces an assessment in less than 200ms for graph pairs with up to 28 nodes, and the results improve as the weight of nodes is higher, as expected. The percentage of incomplete assessments is always less than 5% and lowers as the weight of edges lowers.

The impact of the domain specif data, i.e. the information provided by types and properties was also tested. Although the proposed algorithm is based on structure of the graphs, their nodes and edges, the heuristics use types and properties to distinguish them and improve efficiency. However, it should be noted that the number of types and properties depends the graph-based language and cannot be controlled by the algorithm. As expected, the worst results were obtained with just one or two different types (a single type is equivalent to no types). With three to five types graph pairs of up to an order of 30 are assessed in 50 ms. The incomplete assessments reached 8% for graphs with order 28 with 3 types, but was less than 2% for all orders up to 30 with 4 or 5 types.

The experiments described above were performed with pairs of equivalent graphs to determine the impact of features. In these experiments it was checked that the algorithm found no differences between the graphs. The rest of the experiments were performed with different graphs and was validated that the algorithm recovers the differences introduced by the generator. The algorithm was testes with solutions graphs with sizes up to 30 and attempt graphs with a size variation of up to 8 nodes. The execution time un these assessments is bellow 40 ms, with a tendency to increase with for larger size differences. The number of incomplete assessments is bellow 5% for solution graphs with up to 25 nodes and a different in number of nodes of less than 7.

5 Conclusions and future work

The main contribution of this paper is an algorithm for assessing graphs driven by their structure. It computes both a grade an an explanation, a data object that can be serialized into a natural language text, or used as input for other systems. The assessment algorithm determines the best mapping between nodes in a solution graph and nodes in attempt graph. The mapping is the best in the sense that it maximizes the student's grade.

The algorithm validation ensured its efficiency for connected graphs with up to 30 nodes, which should cover the needs of exercise assessment. It suggests that automatic assessment systems for diagrams can be easily implemented based on this algorithm.

The next step is to validate assessment systems, rather than just the assessment algorithm, by using them with actual graph-based languages and actual students. An experiment is already scheduled for the last month of the current school year with student of a software architecture course.

The motivation for developing this assessment methodology is to blend it with other assessment methodologies, notably the test based assessment used with programming languages. This will require the study of the existing document type definitions for abstract semantic graphs of programming languages used in introductory programing courses such as Java, C/C++, Python and C#, and the existing tools for extracting abstract semantic graphs.

Acknowledgments. This work is financed by the ERDF – European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-037281. Project "NORTE-07-0124-FEDER-000059" is financed by the North Portugal Regional Operational Programme (ON.2 - O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF), and by national funds, through the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT).

References

1. Ali, N.H., Shukur, Z., Idris, S.: A design of an assessment system for uml class diagram. In: Computational Science and its Applications, 2007. ICCSA 2007. International Conference on. pp. 539–546. IEEE (2007)
2. Alur, R., D'Antoni, L., Gulwani, S., Kini, D., Viswanathan, M.: Automated grading of dfa constructions. In: Proceedings of the Twenty-Third international joint conference on Artificial Intelligence. pp. 1976–1982. AAAI Press (2013)
3. Batmaz, F., Hinde, C.J.: A diagram drawing tool for semi–automatic assessment of conceptual database diagrams (2006)
4. Douce, C., Livingstone, D., Orwell, J.: Automatic test-based assessment of programming: A review. Journal on Educational Resources in Computing (JERIC) 5(3), 4 (2005)
5. Shukur, Z., Mohamed, N.F.: The design of adat: A tool for assessing automata-based assignments. Journal of Computer Science 4(5), 415 (2008)
6. Soler, J., Boada, I., Prados, F., Poch, J., Fabregat, R.: A web-based e-learning tool for uml class diagrams. In: Education Engineering (EDUCON), 2010 IEEE. pp. 973–979. IEEE (2010)
7. Thomas, P., Smith, N., Waugh, K.: Automatically assessing diagrams. In: Proceedings IADIS International Conference e-Learning. vol. 2009 (2009)
8. Thomas, P., Waugh, K., Smith, N.: Automatically assessing free-form diagrams in e-assessment systems. In: 1st HEA Aiming for Excellence in STEM Learning and Teaching Annual Conference, Imperial College London (2012)
9. Vachharajani, V., Pareek, J.: A proposed architecture for automated assessment of use case diagrams. International Journal of Computer Applications 108(4), 35–40 (December 2014), full text available