Advances in Cognitive Systems 5 (2017) Submitted 3/2017; published 5/2017

Approximate Graph Matching for Mistake-tolerant Skill Assessment

Melinda Gervasio MELINDA.GERVASIO@SRI.COM
Christian Jones CHRISTIAN.JONES@SRI.COM
Karen Myers KAREN.MYERS@ SRI.COM
SRI International, 333 Ravenswood Ave., Menlo Park, CA 94205 USA

Abstract

This paper presents an approach to automated assessment for online training based on approximate graph matching. The algorithm lies at the core of two prototype training systems that we have built in accord with U.S. Army training materials: one for the use of a collaborative visualization and planning tool, the other for rifle maintenance. The algorithm uses approximate graph-matching techniques to align a representation of a student response for a training exercise with a predefined solution model for the exercise. The approximate matching enables tolerance to learner mistakes, with deviations in the alignment providing the basis for feedback that is presented to the student. Given that graph matching is NP-complete, the algorithm uses a heuristic approach to balance computational performance with alignment quality. A comprehensive experimental evaluation shows that our technique scales well while retaining the ability to identify correct alignments for responses containing realistic types and numbers of learner mistakes.

1. Introduction

Online learning is a large and rapidly expanding market. A widely cited figure puts an estimate of over $100 billion for 2015.[1] One key growth area is online training tools for learning procedural skills such as diagnosing failure of a device or learning how to use a complex piece of software. To date, work in online training systems has focused on teaching the underlying task knowledge. However, proficiency with such skills improves with practice, which in turn requires feedback to foster learning.

We have developed a framework, called Drill Evaluation for Training (DEFT), to support automated assessment of learners as they perform training tasks in online environments (Myers et al., 2013). DEFT uses approximate graph-matching techniques, grounded in edit-distance optimization, to align learner actions with predefined solution models. This alignment provides the basis for generating assessment information with contextually relevant feedback: identifying mistakes, providing hints to help the student complete a task, and suggesting links to relevant training materials. Importantly, our approximate matching approach is tolerant of learner mistakes. This robustness enables assessment of exploratory learning processes rather than forcing learners down fixed solution paths.

While our assessment technology is domain independent, we have applied it within prototype training tools for two significant application domains. One of these assists with training for the

[1] www.forbes.com/sites/tjmccue/2014/08/27/online-learning-industry-poised-for-107-billion-in-2015/#4ff2208d66bc

Command Post of the Future (CPOF)—a collaborative geospatial visualization environment system used extensively by the U.S. Army to develop situational awareness and to plan military operations. In close conjunction with Army trainers, we developed a prototype to assess performance on training exercises drawn from the Battle Staff Operations Course, which provides introductory instruction on CPOF skills. The second is a rifle maintenance application that draws on requirements from a U.S. Army soldier training publication (Greuel et al., 2016).

These two prototypes demonstrated the feasibility of our automated assessment approach in real-world applications. They established the adequacy of our representational framework to capture realistic training problems and their associated solution models. Further, they showed that our approximate matching approach for aligning student responses to solution models works well in practice, enabling the generation of meaningful assessment feedback for mistakes made by students while completing exercises.

The prototypes also showed that, with some algorithm tuning, our automated assessment mechanism performs efficiently on representative problems for those domains. However, given the use of heuristic graph matching at the heart of our assessment algorithm, stronger scalability guarantees are required before fielding a system for deployment: graph matching is an NP-complete problem so can be expensive to solve in the general case. For this reason, heuristic variants are often used (as in our assessment module) that sacrifice optimality for performance gains.

This paper presents our graph-matching approach to automated assessment and a systematic evaluation of it focused on computational performance and alignment quality. We begin by describing our approximate graph-matching approach to automated assessment, detailing the specific matching algorithm used by the assessment module (Section 2). We then present our evaluation (Section 3), which consists of two sets of experiments. The first set evaluates performance under realistic conditions, in terms of both number of actions in an exercise and numbers and types of mistakes. The second set evaluates the algorithm over increasing numbers of errors, to identify thresholds beyond which performance would be unacceptable. We conclude with a discussion of related work (Section 4) and a summary of our results (Section 5).

2. Automated Assessment Approach

DEFT supports online training by providing technology for automated assessment of learned skills. As students work to complete exercises, DEFT performs real-time monitoring to generate a trace of their actions. DEFT then compares these traces to a representation of allowed solutions for the exercise to create assessment information that identifies omissions or mistakes. In addition to notifying the user of these errors, DEFT uses this information to provide guidance in the form of hints to help the student complete a task, and to suggest links to contextually relevant training materials (Myers et al., 2013). Figure 1 shows an example of DEFT's assessment feedback for a student's attempt at the *Clear Rifle* task in the rifle maintenance domain.

There have been decades of work on automated assessment for intelligent tutoring systems. However, most of this work is focused on teaching procedural skills in highly structured domains, such as math or physics. For this type of subject matter, domain models and automated problem solvers can be developed to support assessment via *model tracing* (e.g., Koedinger et al., 1997; VanLehn et al., 2005). In contrast, *example-tracing* tutors (Aleven et al., 2009) assess procedural skills by comparing student actions to a behavior graph that represents all acceptable ways of achieving a task, similar to our approach in DEFT. Because these tutoring systems seek to teach skills, they conduct assessment *in situ* to ensure that the student stays aligned with a validated solution path. In contrast, our target training domains are relatively open-ended, requiring certain critical steps to be performed in some order, but allowing potentially wide variation in the specific steps and the objects on which they operate. For this reason, DEFT focuses on identifying mistakes to assess how well a student has performed a skill overall, without forcing learners to follow a prescribed solution path.

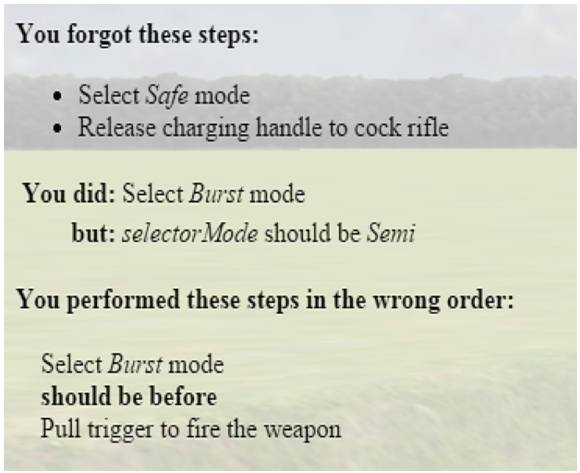

Figure 1. Sample assessment feedback for a rifle maintenance task.

Others have explored similarly challenging domains. For example, the Steve system (Rickel & Johnson, 1999) provides a pedagogical agent that both teaches procedures for operating electro-mechanical devices and provides assistance to learners as they complete training tasks. Steve is designed to provide both interactive critiquing and assistance. As such, similar to most ITS systems, assessment occurs on a per-step basis in order to keep the student on a sanctioned solution path. In contrast, our work focuses on *post hoc* assessment of activities and so does not force learners down predefined solution paths. Rather, our approach supports exploratory forms of learning in which student actions can deviate significantly from validated solution models as they investigate different problem-solving strategies while practicing a skill.

With these requirements in mind, we developed the automated assessment approach in DEFT to support efficient recognition of critical errors in a student response without requiring rigid compliance with a fixed solution and without overly penalizing extraneous but benign actions.

2.1 Alignment Based on Graph Matching

A solution model for a training exercise is composed of one or more *generalized action traces*, each consisting of a sequence of *steps* and *annotations* that specify allowed variations. A step is a *parameterized action*, a *family of actions* (e.g., all the different ways that a delete action can be performed), or a *set of options*, each composed of a partially ordered set of steps. Annotations defined over steps include action ordering and grouping constraints to support specification of,

for example, a partial order between steps, some of which may be specific actions and others of which may be any of a family or actions. Annotations over parameters include type, value, membership, and equality constraints, supporting the specification of, for example, a parameter taking on any of a specified set of alternative values of a particular type or that the output of one action be the same as the input of another.

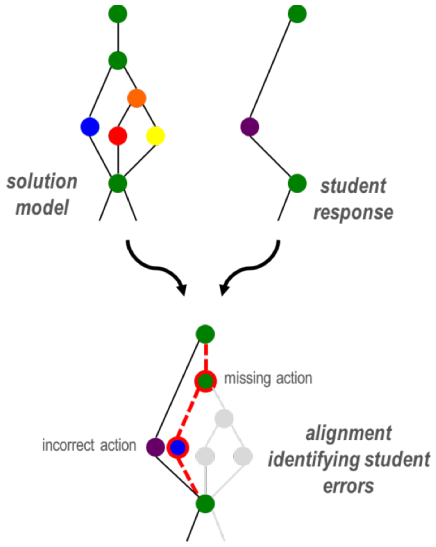

Figure 2. Assessment through graph alignment.

The action traces and accompanying annotations within a solution model define a set of constraints on possible solutions, thus implicitly specifying the valid solution instances for a given exercise. For example, a solution model for the task of cleaning a rifle might include steps for using a rag to clean the upper and lower receivers as well as the buffer assembly and action spring. Annotations can indicate that the cleaning steps may be done in any order, that the upper and lower receivers may be cleaned with a barber brush instead, and that cleaning the buffer assembly and action spring are optional.

The assessment module determines a mapping from a student response to the predefined exercise solution model. We formulate this alignment problem as an approximate graph-matching task, using graph edit distance to rate the quality of the mappings. Graph edit distance measures the cumulative cost of graph editing operations (e.g., deletions, insertions) needed to transform a student response into an instance consistent with the solution model. The intuition is that the lowest-cost alignment corresponds to the solution instance the student is most likely attempting.

To perform assessment through approximate graph matching, the solution model is represented as one or more graphs, each representing a family of possible solutions. Within the graphs, actions and their parameters are nodes, parameter roles are links, and conditions required by the solution are constraints on one or more nodes. The student response is represented similarly as a response graph. Alignment involves finding the lowest-cost mapping between the response and a solution graph, with costs incurred for missing mappings and violated constraints. Figure 2 depicts the process of assessment through alignment between a solution model and a student response. The figure shows a simple case in which there are action nodes only (i.e., no parameter nodes) and links represent ordering constraints. By adjusting the costs for the misalignments, different penalties can be assessed on different deviations. In general, extraneous actions can be assigned zero or minimal cost to encourage exploration and missing actions that are critical may incur higher penalties. For example, in cleaning the rifle, repeating any cleaning step is fine but not cleaning the upper or lower receiver is not.

2.2 Graph Matcher Algorithm

Exact subgraph isomorphism is an NP-complete problem and thus intractable in the most general case; for this reason, the graph matcher that provides the alignment capability within the assessment module performs approximate matching. Specifically, the alignment graph matcher (AGM) employs the A* algorithm (Hart et al., 1968) to search the space of possible alignments between a solution model and a student response.

The AGM generates and maintains a set of partial matches, iteratively selecting a partial match and an unmatched solution node in the partial match to expand next, then extending the set of partial matches with candidate matches for that node. It chooses as the next partial match to expand the match with the lowest estimated total cost $f(m) = g(m) + h(m)$, where $g(m)$ is the cost of the partial match so far and $h(m)$ is the estimated cost for the remaining unmatched nodes. Search terminates when one of the following conditions holds:

- a full match is found whose cost is lower than the estimated total cost of any remaining partial match;
- there are no more partial matches to expand; or
- a designated search limit on the number of partial matches expanded is reached.

Because the AGM only expands partial matches by adding additional matches (i.e., it does not consider expansions that delete existing matches), the search space is acyclic and so with an admissible heuristic for estimating remaining cost, A* would be guaranteed to find an optimal solution. However, finding such a heuristic that can be computed efficiently for our graph matching problem is challenging: simple heuristics such as assuming all remaining nodes will be matched provide too loose a bound to effectively direct search while heuristics based on true cost devolve to exhaustive search. Indeed, initial implementations of the AGM that used these simple heuristics proved to be computationally infeasible in practice. Thus, the AGM employs the following approach to estimate remaining cost instead.

Recall that a solution graph consists of action nodes and parameter nodes, with the parameters of an action linked by parameter role edges to their actions. In expanding a partial match, the AGM decides on which unmatched solution node to attempt to match next by selecting the *heaviest* action node, where node weight is defined as the sum of the costs of not matching the action node and all its parameter nodes, and of not satisfying all associated constraints. This greedy heuristic is based on the intuition that matching the heaviest nodes first will tend to lead to the lowest-cost alignments more quickly—similar to the *fail-first heuristic* used in constraint optimization for which the variable with the fewest options is instantiated next (Haralick & Eliot, 1980). The primary focus is on matching action nodes since the parameter nodes attached to an action in a solution must be matched to the parameter nodes attached to the matching action in the response.

To compute the estimated remaining cost $h(m)$ of a partial match m, the AGM relies on an *a priori* analysis of the solution against the response, in which it compares the nodes to be matched and the constraints to be satisfied in the solution against the space of possible node matches in the response. The analysis is based on the idea of measuring *node shortfall*. Intuitively, if there are n nodes of a particular type in the solution and m nodes of a matching type in the response, and $m <$

n, then at least *n – m* nodes in the solution will go unmatched. Thus, any match will incur the cost of deleting the unmatched nodes as well as the cost of all constraints associated with that node. By assuming that the cheapest of the solution nodes will go unmatched, we can calculate a best-case match cost.

A similar analysis is performed on the constraints. Given an *n*-ary constraint, the *n* variables must be satisfied by different nodes (i.e., there is no constraint that expresses a relation over the same object). Further, given *m* instances in the solution of the same type of constraint, each one must be satisfied by a different combination of nodes; otherwise, there would be duplicate constraints in the solution. Thus, if there are only *j* possible combinations of nodes that could potentially satisfy *k* constraints, and $j < k$, then there will be $k - j$ constraints left unsatisfied and any match will incur the corresponding cost. By assuming that the cheapest of these constraints will remain unsatisfied, we can calculate a best-case match cost.

The AGM uses this prior analysis to estimate the remaining cost for any partial match by first filtering out the nodes that have already been matched and the constraints that have been satisfied. Then, it applies the same analysis on the remaining nodes and constraints using the precomputed costs to estimate the best-case cost for the remaining unmatched nodes. Finally, to break ties, the AGM uses additional heuristics to prefer closer temporal matches (i.e., nodes in the response and the solution that are in about the same location) and contiguous matches.

After an unmatched action node in a partial match is selected for matching next, all possible mappings between solution and response are generated for the action node and all its attached parameter nodes. The selected action node in the solution is matched to action nodes in the response, and the associated parameter nodes in the solution are similarly matched to parameter nodes in the response, with constraint violations accounted for in the cost of the partial match. The new mappings result in expanded partial matches that are added to the queue according to their estimated full cost *f(m)* and search continues.

3. Graph Matcher Evaluation

We conducted two sets of experiments to evaluate the performance of the AGM. The first set evaluated performance under realistic error conditions. The second set investigated performance under increasing numbers of specific types of errors, as a means to identify scalability limits.

3.1 Experimental Design

The experiments were conducted on synthetically generated responses created through controlled modifications to a correct solution. Access to actual users in our application domains is limited. Creating synthetic data in a principled manner was necessary to enable the comprehensive and systematic evaluations that we sought to conduct.

Our process for generating synthetic responses is as follows. First, a solution model with desired characteristics (e.g., fully/partially ordered actions, unique/repeated actions, unique/repeated parameter values) is created from a randomly generated model of possible actions. Next, a valid instance of the solution model is generated by selecting parameter values for each action that satisfy the parameter constraints and sequencing the actions in accord with the ordering constraints. From this solution instance, erroneous responses are created through

perturbations that introduce unmatchable nodes in the solution or in the response, and/or that break constraints. Each perturbation introduces specific errors; ideally the AGM would find exactly those errors during alignment. By controlling the perturbations, we can evaluate performance against different types and numbers of errors.

3.2 Metrics

We consider two metrics: *computational performance* and *alignment quality*. To evaluate computational performance, we measure CPU time and number of expansions (i.e., the number of partial matches expanded by the AGM until an alignment is returned). To evaluate alignment performance, we measure *precision* and *recall* on the errors introduced by the perturbations. Precision measures the ability of the AGM to predict true errors while recall measures the ability to find the expected errors. More specifically, let

- *misalignment*: a missing node, extra node, or violated constraint
- *found misalignment*: a misalignment identified by the AGM
- *actual misalignment*: a (real) misalignment introduced by a perturbation
- *true positive* (*TP*): a found misalignment that is also an actual misalignment
- *false positive* (*FP*): a found misalignment that is not an actual misalignment
- *false negative* (*FN*): an actual misalignment that is not a found misalignment

Then for a response *r*: $precision(r) = TP / (TP + FP)$ and $recall(r) = TP / (TP + FN)$. Precision/recall over a set of responses *R* is the average over the responses *r* in *R*.

3.3 Experiment 1: Realistic Errors

3.3.1 Overview

The objective for the first experiment was to evaluate AGM performance against realistic student errors. Drawing from the literature on human errors (Trinh et al., 2009; Trafton et al., 2011), we identified and investigated nine types of errors that occur in the performance of procedural tasks; these are summarized in Table 1. Each error type can be generated through a set of *perturbations* to the student response, summarized in Table 2. Five of these perturbations involve actions while one involves parameters. Each perturbation has the potential to introduce one or more problems that ideally would be identified by the AGM.

We conducted two sets of trials. One explored performance in the presence of a single error. The second investigated performance for multiple errors of different types. The experiment leveraged a base set of 20 different actions, each having one to three parameters. We randomly generated solution models consisting of 20 arbitrary instantiations of arbitrarily chosen actions. Thus, solutions generally contained some repeated actions and parameter values. For every action, an ordering constraint with the previous action was added with 80% probability. In comparison, the largest response for training exercises in the rifle maintenance domain had 16 actions.

Table 1. Summary of errors investigated and corresponding perturbations.

Error Type	Description	Corresponding Perturbation
perseveration	repeatedly executing the same action	immediately repeat an action one or more times
reversal	switching the order of execution of two actions	switch the order of two consecutive actions
jump forward	skipping ahead	delete a contiguous segment (one or more consecutive actions)
jump backward	going back	immediately repeat a segment
initialization	forgetting preparatory actions (typically before the main task)	delete a segment from the start of a response
post-completion	forgetting clean-up actions (typically after the main task has been completed)	delete a segment from the end of a response
anticipation	performing actions too early, leading to having to repeat them later	pick a segment and insert a repetition of it anywhere earlier in the sequence
capture error (action)	incorrect (but similar) action executed instead	replace an action with some other action
capture error (parameter)	correct action executed on wrong object	replace the parameters of an action with some other parameters

Table 2. Perturbations and corresponding misalignments.

Perturbation	Misalignment
deleted action	missing actions, violations of associated constraints
different action	missing action and extra action, violations of associated constraints
inserted action	extra action
switched actions	violations of associated ordering constraints
moved action	violations of associated ordering constraints
different parameter value	violations of associated value and equality constraints

3.3.2 Results

Table 3 summarizes the results for individual errors. For each error type in Table 1, we generated 30 perturbations of a baseline solution with 20 actions. We ran the AGM over each perturbation along with the baseline solution to evaluate AGM performance in determining an alignment. Here, the number of violated constraints in the response (i.e., misalignments) provides a measure of problem difficulty. In all cases, the AGM found the expected alignments quickly (within 30 msecs) and with little search (no more than 30 expansions). It also correctly identified all mistakes, i.e., precision and recall were perfect.

Table 3. Computation times (in msecs) and number of expansions for (a) single error cases (over trials with 20 incorrect responses) and (b) the correct response.

	Incorrect									Correct	
	# Violations			# Expansions			CPU Time			CPU Time	# Exp.
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg		
Perseveration	1	2	1.6	14	15	14.3	21.4	30.5	25.1	30.3	14
Reversal	1	1	1.0	10	10	10.0	14.4	16.3	15.1	15.0	10
Jump forward	4	10	7.1	17	18	17.4	18.9	24.8	20.9	20.4	17
Jump backward	1	2	1.6	10	12	11.0	15.2	20.2	17.1	15.3	10
Initialization	3	12	8.7	16	17	16.5	17.4	19.5	18.4	17.8	16
Post-completion	3	10	6.2	13	13	13.0	13.7	14.3	13.9	14.4	13
Anticipation	2	2	2.0	12	27	18.4	16.5	24.7	20.2	15.0	12
Param capture	1	3	2.0	13	13	13.0	14.7	16.6	15.2	14.8	13
Action capture	4	13	8.3	14	17	14.7	16.3	19.5	17.5	16.9	14

For combinations of error types, we conducted 30 trials each on 30 different randomly generated solutions, introducing two to four errors per trial. Errors were generated randomly from a 60:40 distribution over *slips* and *mistakes*, where a slip manifests as one of the nine errors in Table 1 and mistakes as one of the first seven.[3]

Figure 3 displays the results for the 30 trials. The top graph shows computational metrics (# of violated constraints, # of expansions, CPU in msecs), for which lower is better; the bottom graph shows alignment quality metrics (precision, recall), for which higher is better. As can be seen, computational performance remains good with a maximum of 32 msecs per run. However, precision and recall are sometimes imperfect. A closer look at the imperfect alignments reveals that the AGM sometimes found an alternative but arguably just as valid alignment. For example, if the erroneous response is generated by deleting one (unique) action and then replacing another with a different instance of the unique one, our scoring scheme would expect two missing actions and an extra action but the AGM might instead find one missing action and violated ordering constraints.

3.4 Experiment 2: Scalability Limits

The second experiment explored scalability limits of our approach. We started with a baseline solution model consisting of a partially ordered set of unique one-parameter actions. We investigated the effects of specific types of errors by incrementally adding perturbations of that type to the solution. For example, to investigate the effect of missing actions, we ran successive

[3] In the literature on human errors, *mistakes* derive from conceptual misunderstandings while *slips* derive from incorrect application of accurate conceptual knowledge. Reason (1991) found error frequencies to be 61% skill-based, 27% rule-based, and 11% knowledge-based, where the latter two (*mistakes*) have the same manifestations in our setting. Our evaluation is based only on the manifestations and not the causes of the errors, leading to the 60:40 distribution used in the experiment.

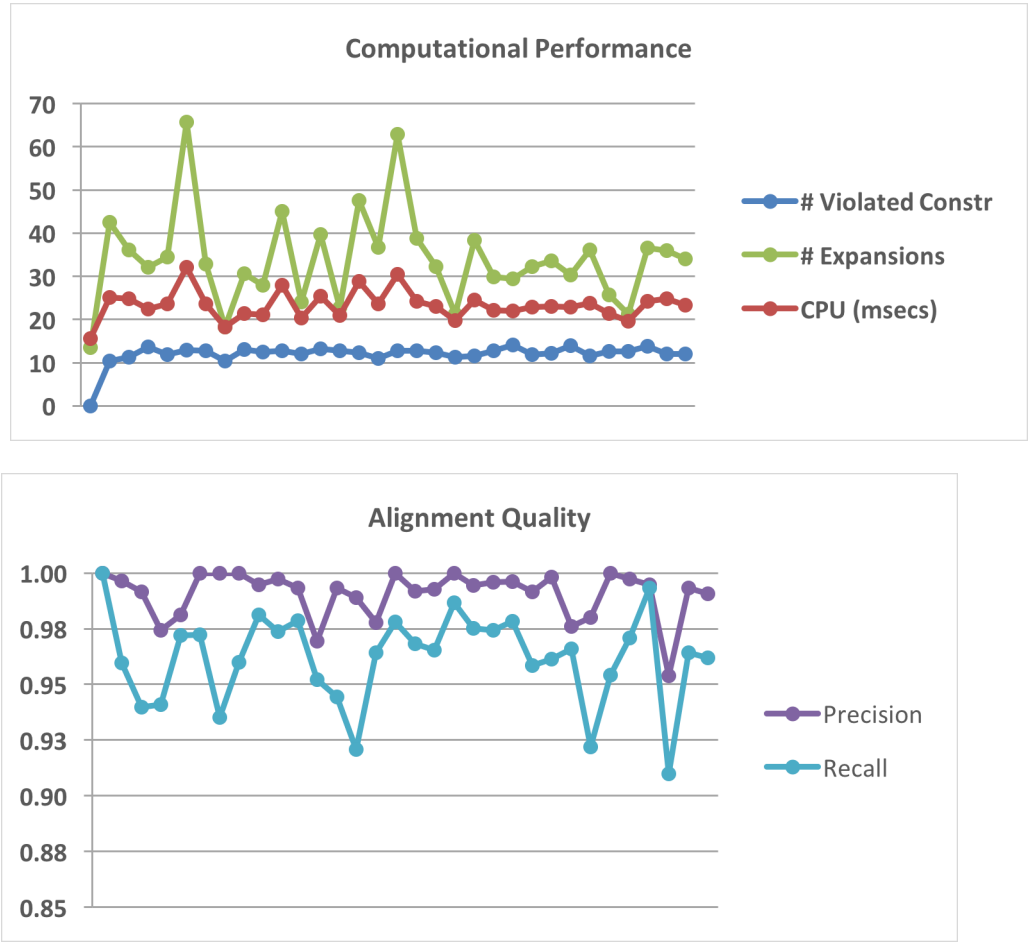

Figure 3. Results for aggregate heterogeneous mistakes.

cases with 0–100 deleted actions. For most perturbation types, we ran 100 cases; for other types, we were limited to 50 because of the nature of the change (e.g., switched actions were done in pairs). Because the AGM's search space grows with the number of possible matches, we also explored the case of different actions in which we varied the perturbations as to whether they introduced repeated actions and/or parameter values in the erroneous response.

Figures 4 and 5 show results for 30 trials each on 30 different randomly generated baseline solutions. The x-axis plots increasing numbers of perturbations, with successive erroneous responses building upon the previous response. In all cases, the AGM finds the correct alignments, so the display omits precision and recall (i.e., they are uniformly perfect).

Figure 4 shows results for the case in which solutions do not contain repeated actions and erroneous responses do not introduce repetitions. In this case, computational performance remains

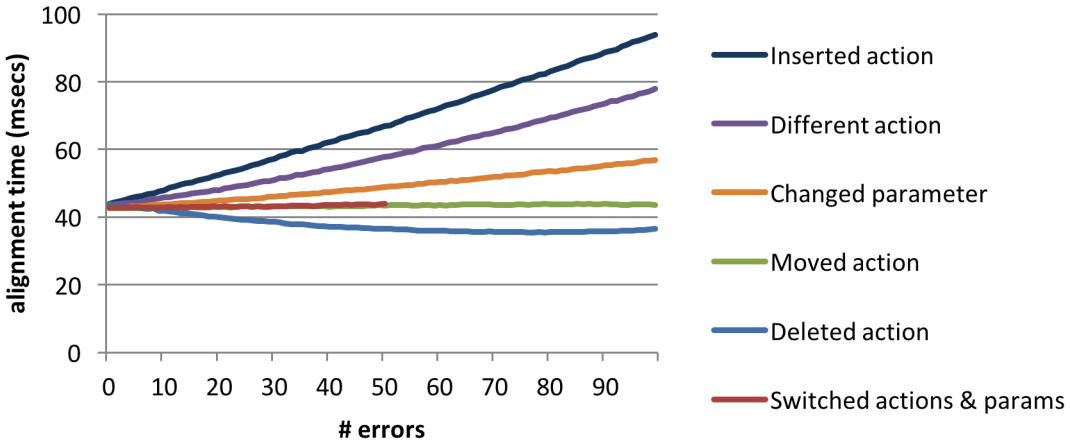

Figure 4. Impact of increasing # of perturbations without repetitions.

excellent throughout for all perturbation types (below 100 msecs). Computation time generally increases linearly with the number of perturbations; for the case of action deletion, however, computation time drops with the length of the response since the corresponding matching problem is easier.

Figure 5 explores the impact of repetitions for both the "different action" perturbation (where successively more actions are replaced by incorrect ones) and the "additional action" perturbation (where successively more unnecessary actions are inserted). In both cases, repeated parameters have negligible effect and repeated actions have modest effect. While the combination of repeated actions and parameters shows exponentially increasing runtimes, performance even at the pathological extremes of 50+ errors for the "different action" condition (i.e., over half of the actions are wrong) remains acceptable (below 4 seconds).[4] Similar results hold for the "additional action" condition.

4. Related Work

There is a long line of work on graph matching in pattern recognition, with successful application in image and video analysis, document processing, and biological and biomedical applications (Conte et al., 2004). Neuhaus et al. (2006) investigate a number of suboptimal A* variants for computing graph edit distance and show dramatic computational gains with some sacrifice in accuracy when used in a nearest-neighbor image classification task.

Conformance checking for workflows involves determining whether a given recorded trace of actions is consistent with a model of allowed workflows. (Adriansyah et al., 2011) represent a

[4] Martin and Corl (1986) reported finding no reliable effect on performance for problem-solving tasks when response delays were below five seconds.

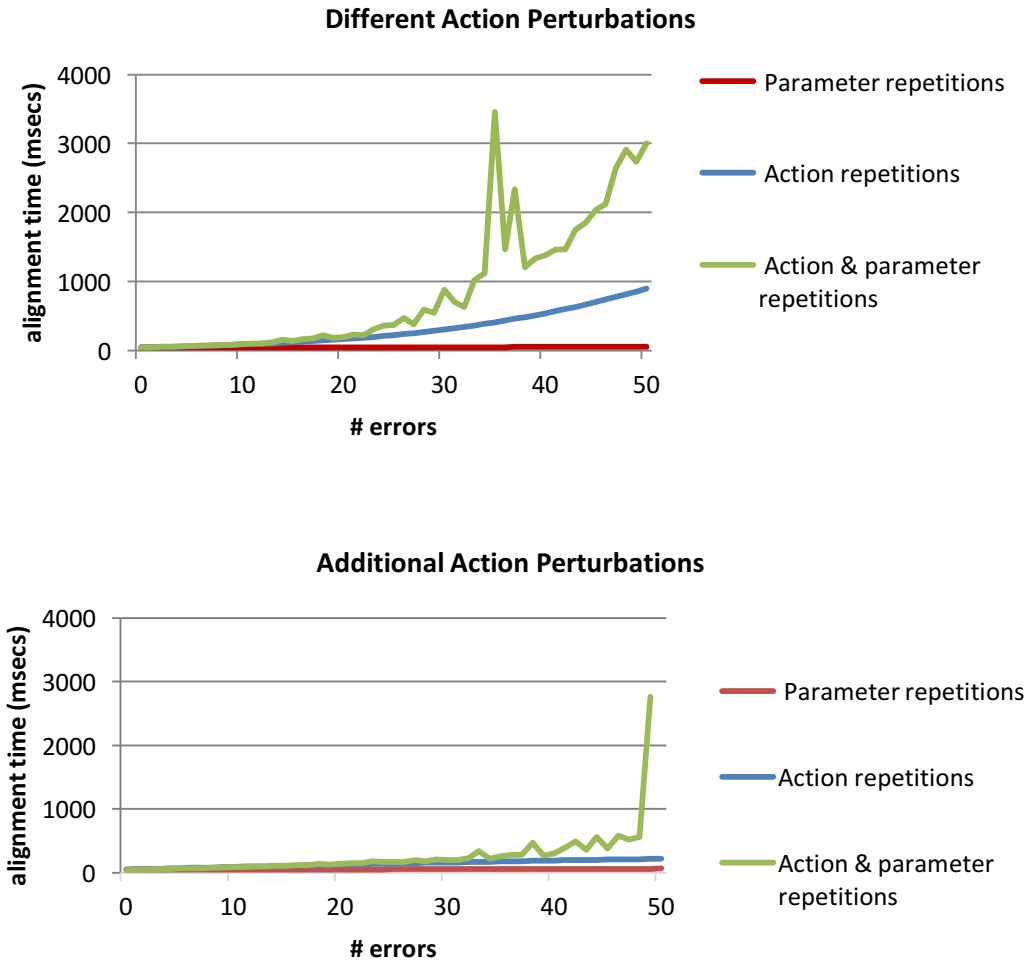

Figure 5. Impact of increasing # of perturbations with repetitions.

workflow model as a set of actions along with constraints on those actions, similar to our formulation of exercise solution models. Their conformance checker uses an A* algorithm that seeks to minimize insertion and deletion of actions.

5. Conclusions

This paper presents an approximate graph-matching approach to the automated assessment of procedural skills. Experimental results show that the approach is scalable and can handle realistic numbers and types of errors. The graph-matching approach is tolerant to learner mistakes,

enabling assessment of exploratory learning processes rather than forcing learners down fixed solution paths. The use of edit distance to assess match quality lets us adjust alignment behavior to assess penalties consistent with the significance of different errors.

While our graph-based approach to assessment provides the flexibility needed in our target domains, exact subgraph matching is intractable so we use a heuristic approach instead. Prototype applications of our approach for automated assessment in real-world domains showed that it worked well on representative problems. The experimental studies described in this paper were motivated by the goal of determining whether our approach would scale to real-world training conditions. Our experiments showed that the AGM performs well under realistic error conditions. It finds the misalignments corresponding to the different types of errors accurately and quickly, considering both computational time and the number of expansions performed during search.

A major impediment to deployment of automated assessment capabilities of this type is the high cost associated with creating the solution models that drive assessment. In related work, we have developed an approach to model authoring rooted in end-user programming techniques (Myers & Gervasio, 2017). In particular, the authoring process consists of a combination of demonstrating solutions augmented by mechanisms for generalizing from demonstrations to a comprehensive solution model. With this approach, content creation no longer requires deep understanding of the knowledge representation and inference mechanisms within the assessment module. As such, content can be created by domain experts after a modest amount of training.

Acknowledgments

This material is based upon work supported by the United States Government under Contract No. W911QY-14-C-0023. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Government. The authors thank Chris Greuel for his activities in support of this work.

References

Adriansyah, A., van Dongern, B., & van der Aalst, W.M.P. (2011). Conformance checking using cost-based fitness analysis. In C.H. Chi and P. Johnson (Eds.), *IEEE International Enterprise Computing Conference*.

Aleven, V., McLaren, B., Sewall, J. & Koedinger, K. (2009). A new paradigm for intelligent tutoring systems: example-tracing tutors. *International Journal of AI in Education*, *19*(2).

Conte, D., Foggia, P., Sansone, C., & Vento, M. (2004). Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and AI, 18*(3), 265–298.

Gervasio, M., Myers, K., & Wessel, N. (2017). Demonstration-based solution authoring for skill assessment. *Proc. of the Fifth Annual Conference on Advances in Cognitive Systems*.

Greuel, C., Myers, K., Denker, G., & Gervasio, M. (2016). Assessment and content authoring in semantic virtual environments. *Proc. of the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC).*

Haralick, R., & Elliott, G. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence, 14*.

Hart, P., Nilsson, N., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics, 4*, 100–107.

Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of AI in Education*, *8*, 30–43.

Martin, G.L. & Corl, K.G. (1986). System response time effects on user productivity. *Behaviour and Information Technology*, 5(1), 3–13.

Myers, K. Gervasio, M., Jones, C., & Keifer, K. (2013). Drill evaluation for training procedural skills. *Proc. of the 16th Intl. Conf. on AI in Education*.

Neuhaus, M., Riesen, K. & Bunke, H. (2006). Fast suboptimal algorithms for the computation of graph edit distance. *Joint IAPR International Workshops on STPR&SPR*.

Reason, J. (1991). *Human Error*. Cambridge University Press.

Rickel, J. & Johnson, W. L. (1999). Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. *Applied Artificial Intelligence, 13*.

Trafton, J. G., Altmann, E. M., & Ratwani, R. M. (2011). A memory for goals model of sequence errors. *Cognitive Systems Research, 12*, 134–143.

Trinh, T.-H., Buche, C., Querrec, R., & Tisseau, J. (2009). Modeling of errors realized by a human learner for training. *International Journal of Computers, Communications & Control, 1*, 73–81.

VanLehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R., Taylor, L., Treay, D., Weinstein, A., & Wintersgill, M. (2005). The Andes physics tutoring system: lessons learned. *International Journal of AI in Education*, *15*(3).