Comparison of Algorithms for Automatically Building Example-Tracing Tutor Models

Rohit Kumar Matthew E. Roy R. Bruce Roberts John I. Makhoul
Raytheon BBN Technologies
Cambridge, MA, USA
{ rkumar, mroy, broberts, makhoul } @ bbn.com

ABSTRACT

In our recent work, we have proposed that multiple behavior demonstrations can be automatically combined to generate an Example-Tracing Tutor model. In this paper, we compare four algorithms for this problem using a number of different metrics for two different datasets, one of which is publicly available. Our experiments show that these four algorithms are complementary to each other in terms of their performance along the different metrics. These findings make a case for incorporating multiple algorithms for building behavior graphs into authoring tools for Intelligent Tutoring Systems (ITS) that use behavior graphs.

Keywords

Tutor Models, Example-Tracing Tutors, Behavior Graphs, Authoring, Automation, Algorithms, Metrics

1. INTRODUCTION

Conventionally, Example-Tracing Tutors [1] are developed in three stages by trained domain experts: (1) User Interface development, (2) Behavior demonstration, (3) Generalization and annotation of the behavior graph. Recently, we proposed [2] that the effort involved in Stage 3 of this process can be significantly reduced by using algorithms that can automatically create a generalized behavior graph from multiple demonstrations.

Automation of tutor model development has been explored in different contexts using completely automated methods as well as augmentation of authoring tools. Barnes and Stamper [3] proposed a method that uses existing student solutions to generate hint messages for the Logic Proof tutor. Recently, Eagle et al. [4] have used clustering of interaction network states as an approach to the same problem. In the context of knowledge-tracing and example-tracing tutors, McLaren et al. [5] proposed the use of activity logs from novice users to bootstrap tutor model development. They developed software tools that integrate access to novice activity logs with tutor authoring tools.

In the next section, we briefly outline four algorithms for automatically generating behaviors graphs. In Section 3, we will present experiments using two datasets, to compare these algorithms along a number of metrics that measure desirable characteristics of tutor models.

2. ALGORITHMS

2.1 Behavior Graphs and Demonstrations

Behavior graphs are directed graphs. The nodes in a graph correspond to valid solution states. Non-terminal nodes represent partial solutions. Edges in the graph represent events, some of which are correct and lead to the next state while others are incorrect and lead back to the same state. Edges are annotated with the conditions that an event must meet to traverse the edge. Behavior graphs may also include unordered groups. As the name suggests, states within an unordered group may be traversed in any order. Constituents of the behavior graph (i.e. nodes, edges, groups) may be associated with a number of annotations based on the educational application.

On the other hand, behavior demonstrations are captured as a sequence of user interface (UI) events. Each event is represented as a 2-tuple $e_i = (u_i, \mathbf{d}_i)$ that includes an identifier u_i of the UI element and data $\mathbf{d}_i$ associated with the event. Note that each behavior demonstration implicitly represents a behavior graph where the nodes in the graph correspond to the state of completion of each event in the demonstration. Such a behavior graph does not generalize to learner behaviors beyond those that are exactly identical to the demonstration. Automatic Behavior Graph Generation (ABGG) algorithms utilize multiple demonstrations of solutions of a problem to generate a behavior graph that can serve as a tutor model for the problem.

2.2 Algorithm 1: Interaction Network

The baseline algorithm used in our work combines the individual behavior graph corresponding to available demonstrations by merging identical nodes and edges in a sequential order. When a non-identical edge is found, a new branch is created in the graph. The resulting behavior graph is an interaction network which has been used in prior work [4] [6]. All paths in the behavior graph generated by this algorithm are assumed to be correct paths i.e. this algorithm is incapable distinguishing between correct and incorrect actions by the learner. While the behavior graph generated by this algorithm is more general than any individual demonstration used to create the graph, no unseen paths are generated. Furthermore, the number of nodes and edges created by this algorithm is fairly large, which makes the annotation of such graphs difficult for problems with many UI elements.

2.3 Algorithm 2: Heuristic Alignment

Our next algorithm, shown in Table 1, utilizes two characteristics of behavior demonstrations. First, if two or more events in a demonstration have the same element identifier u_i, the latter event likely corresponds to a correction of the data value input in the former events. Second, if we assume that there is one and only one correct solution sequence through the UI elements, we can transform the problem of generalizing behavior demonstrations to that of finding the optimal sequence of states through the UI elements.

This research was funded by the US Office of Naval Research (ONR) contract N00014-12-C-0535.

Table 1. Algorithm 2 (Heuristic Alignment)

Stage 1. Compute Retracted Demonstrations
- For each demonstration D
  - For each retracted event e = (u, **d**)
    1. e_{target} = last event in D s.t. $e_{target} \rightarrow u = e \rightarrow u$
    2. Add $e \rightarrow \mathbf{d}$ to $e_{target}.\mathbf{d}_{wrong}$
    3. Remove e from D

Stage 2. Find Sequence of States
- For each unique identifier u
  1. $\mathbf{p}_u$ = set of positional indices of events s.t. identifier = u
  2. $mode_u = mode(\mathbf{p}_u)$
- Sequence states (s_u) corresponding to each element identifier (u) in increasing order of their $mode_u$

Stage 3. Generate Edges
- For each state s_{u*}
  1. Generate correct edge for each unique **d** for events s.t. identifier = u*
  2. Generate wrong edge for each unique entry in $\mathbf{d}_{wrong}$ for events s.t. identifier = u*

Stage 4. Identify Unordered Groups
- For each pair of adjacent states (s_{u1},s_{u2})
  1. if $|\cap(\mathbf{p}_{u1},\mathbf{p}_{u2})| > \sqrt{|\text{demonstrations}|}$, group s_{u1}, s_{u2}

2.4 Algorithm 3: Center Star Alignment

Note that Stage 2 of the previous algorithm is, in effect, aligning the multiple demonstrations. The Center Star Algorithm can be used to perform this alignment. Algorithm 3 uses the Center-Star Alignment between the retracted demonstrations. Similar to algorithm 2, a new state is generated for each position in the aligned demonstrations. However, since we obtain the alignment using the Center Star algorithm, the second assumption made by algorithm 2 is not necessary, which can lead to multiple states with the same element identifiers. This allows algorithm 3 to generate alternate paths.

2.5 Algorithm 4: Combining Multiple Paths

Algorithm 4 considers ABGG as the process of finding multiple paths in a directed graph. A first order transition matrix obtained from the demonstrations represents a directed graph. Specifically, the longest (non-repeating) path in this directed graph is the most likely path through the UI elements based on the demonstrations. While the problem of finding longest paths in general graphs is known to be NP-hard, in our approach, we employ an exponential time longest path finding algorithm within bounds of the number of UI elements and uses a transformed transition matrix to find multiple shortest paths. The transform changes the weight of each valid edge of the directed graph to row normalized inverse. We merge all the paths found to we construct a behavior graph similar to the process of constructing an interaction network. The algorithm uses Stage 1 and Stage 4 of algorithm 1.

2.6 Discussion

As mentioned earlier, incremental addition of demonstrations to generate interaction networks does not identify incorrect input data values. Using the assumption about retracted events, the other three algorithms are able to identify incorrect inputs. Johnson et al. [6] used a similar assumption in their work on reducing the visual complexity of interaction networks. We notice that the algorithms 2 and 3 are complementary in terms of their ability to find alternate paths and unordered groups. Algorithm 4 on the other hand offers both of these abilities. In the next section, we will discuss the performance of all of these algorithms in terms of quantitative metrics

None of the algorithms discussed in this paper are capable of discovering data values beyond those seen in the training demonstrations. This type of generative ability is particularly useful for learning tasks, such as language learning, where a large number of different inputs may be expected from the learners. In our ongoing work, we want explore the use of grammar induction techniques to learn regular expressions from correct and incorrect data values for each state.

3. EVALUATION

3.1 Datasets

We use two collections of behavior demonstrations/traces to evaluate the performance of the four algorithms described earlier. The first dataset (referred to as the *BBN dataset*) comprises of five physics problems. Nine subjects spent upto one hour each to create demonstrations of the five problems. All nine subjects were able to complete demonstrations of three problems. Six subjects completed the fourth problem and only four completed the fifth problem. Additionally, we used three Assistments datasets accessed via DataShop [7] to form our second collection of behavior demonstrations. This publicly shared large dataset comprises a total of 683197 traces and 1905672 events for 3140 problems. We filtered these datasets to use only problems that had six or more traces and had at least two UI elements.

3.2 Metrics

Metrics used in our evaluation are discussed in detail in our prior publication [2]. These metrics are categorized by the desirable characteristics of automatically generated behavior graphs they measure.

- **Readability/Maintainability:** The conciseness of a graph can be measured using the number of nodes and edges in the graph. Compression ratio measures the rate at which an algorithm is able to reduce demonstration events into behavior states (i.e. nodes) by finding similarities between events.
- **Completeness:** We use the rate of unseen events in held out demonstrations as a metric to measure the completeness of our automatically generated behavior graphs.
- **Accuracy:** Edge accuracy measures the percentage of Correct & Incorrect edges that were accurately classified by the algorithm. Error rate is a frequency weighted combination of edge accuracy that measures the fraction of learner events that will be inaccurately classified by the automatically generated behavior graph.
- **Robustness:** Branching factor is the average number of data values available at each UI element. A large branching factor indicates the capability to process a large variety of learner inputs at each state. Also, the number of unordered groups and the size of unordered groups are indicative of flexibility a graph affords to learners to explore the solution paths of a problem.

3.3 Experimental Design

We use two different experimental designs for the two datasets. Since the BBN dataset is comprised of a small number of demonstrations per problem, we use all available demonstrations for training and report only the metrics that can be derived from the graphs and the training demonstrations. Since a large number of traces are available for the problems in the Assistments dataset,

Table 2. Averaged metrics for the graphs generated for the problems in the BBN & Assistments (Math) dataset
*indicates significant ($p < 0.05$) difference with other algorithms for the same dataset

Metrics ▼	Algorithm 1		Algorithm 2		Algorithm 3		Algorithm 4	
	BBN	Math	BBN	Math	BBN	Math	BBN	Math
#Nodes	144.8	79.1*	**32.4**	**5.4**	37.0	6.0	**32.4**	6.6
#Correct Edges	160.4	147.9*	**70.0**	**12.8**	97.0	18.3	71.4	17.5
#Incorrect Edges			17.2	24.2*	19.8	33.4*	**5.0**	**19.5***
Compression Ratio	1.8	6.7*	**7.3**	**77.3***	6.3	66.8*	**7.3**	60.2*
% Accurate Correct Edges	76.7	39.1*	77.0	42.2	66.2	42.6	**82.8**	**44.1***
% Accurate Incorrect Edges			99.5	**99.9***	99.5	97.5*	**100.0**	99.5*
Training Error Rate	15.5	51.3*	**7.6**	25.2*	13.0	17.7	**7.6**	**17.4**
Heldout Error Rate		42.7*		23.4*		16.0		**15.6**
% Training Unseen Events	**0.0**	**0.0**	**0.0**	10.5*	4.4	2.2*	8.1	6.7*
% Heldout Unseen Events		**10.1***		19.0*		11.5*		13.8*
Branching Factor	1.2	2.2*	3.1	11.1*	**3.4**	**12.6***	2.7	8.5*
#Groups			1.6	**0.5***			**2.6**	0.02*
Avg. Group Size			**3.3**	**1.8***			2.8	0.04
% Group Coverage			17.7	**30.6***			**31.3**	0.5

we use a three-fold cross validation design to split the available traces into three different training and held out sets. Reported metrics are averaged over each split.

3.4 Results

3.4.1 BBN Dataset

Table 2 shows performance results for the four algorithms on the two datasets. As expected, the interaction networks comprise of a large number of nodes and edges that lead them to have significantly ($p<0.01$) lower compression ratio. Algorithms 2 (Heuristic Alignment) and 4 (Multiple Paths) are able to achieve the highest compression consistently for all five problems.

On the accuracy metrics, Algorithm 4 outperforms the other algorithms on average. However, it is significantly better ($p<0.001$) than Algorithm 1 and 3 on the incorrect edge accuracy metric. Furthermore, the high accuracy for incorrect edges for two of the three algorithms that use the retracted demonstrations partly validates the underlying assumption made by these algorithms.

In contrast to the accuracy metrics, alignment based algorithms (2 and 3) outperform the multiple paths algorithm (4) on achieving a higher branching factor. The frequency based pruning underlying the selection of multiple paths in algorithm 4 leads to the elimination of certain novel edges. Based on the performance of these algorithms on the edge accuracy metrics we see many of these novel edges are likely to be inaccurate due to limited evidence for their classification in the training demonstrations. While the algorithms complement each other, Algorithm 4 seems to be a potential candidate for optimal tradeoff between the different metrics.

In terms of metrics based on unordered groups in a graph, we find that algorithm 4 leads to a larger fraction of nodes (31%) to be included in unordered groups. Finally, we see that pruning significantly degrades the performance of Algorithm 4 on percentage of unseen events i.e. completeness. Since interaction networks losslessly embed all events observed in the training demonstration, their performance on this metric is guaranteed to be flawless. In the next section, we will compare this result to their performance on held out demonstration sequences.

3.4.2 Assistments Dataset

The performance of the algorithms on the Assistments (Math) dataset is also shown in Table 2. Largely, the results on this dataset agree with the results on the BBN dataset. Algorithm 2 (Heuristic Alignment) outperforms all other algorithms on three of the readability metrics. Unlike the BBN dataset, the average compression ratio for Algorithm 2 is significantly better than the other algorithms including Algorithm 4 (Multiple Paths).

Algorithm 4 significantly outperforms the other algorithms on three of the accuracy metrics. Because of their lossless nature, Interaction Networks (Algorithm 1) performs the best on Completeness metrics (% unseen events) as was the case with the BBN dataset. However, we find evidence of over-fitting of the algorithms to training data on this metric as indicated by the approximately 9% higher rate of unseen events for held out demonstrations for all the algorithms.

While the results on the branching factor metrics of the Assistments dataset are consistent with the BBN dataset, Algorithm 2 outperforms Algorithm 4 on the metrics based on the unordered groups. Because Algorithm 2 identifies unordered groups that are larger in size than Algorithm 4, the groups found by the Heuristic Alignment algorithm have a higher coverage of the generated graphs, especially in the Assistments datasets where the number of UI elements is relatively small.

Figure 1 further explores the tradeoff between the key metrics for larger number of traces (i.e., more training data) in Figure 1a and increasingly complex problems (i.e., more UI elements) in Figure 1b. Algorithm 1 does not scale well on readability metrics (Compression Ratio). The algorithms demonstrate stability in accuracy and completeness performance with increasing problem or data complexity. Algorithms 3 and 4 can produce a consistently low error rate despite increasing complexity. The rate of unseen events reduces by over 60% (relative) for a 10-fold increase in training data. This is also

Figure 1a. Algorithm performance for different number of training traces

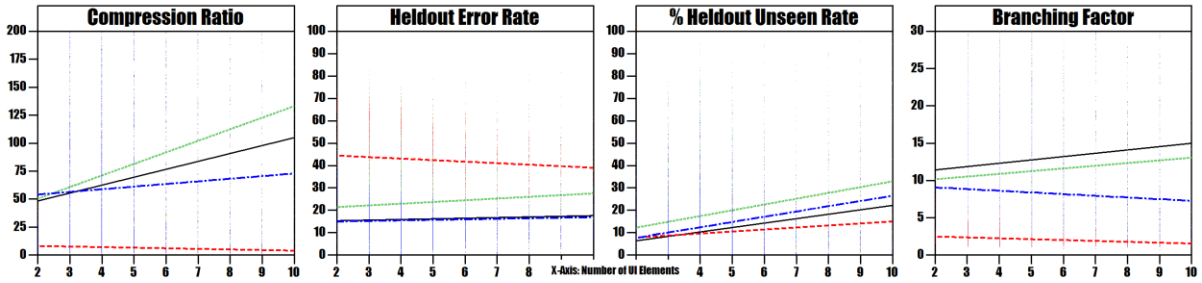
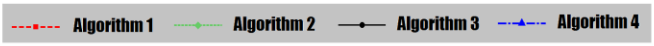


Figure 1b. Algorithm performance for different number of UI elements in a problem



evidenced in the BBN dataset if we compare problems 1, 2 and 6 which have more data than problems 10 and 15. Finally, as is often the case with data-driven approaches, model robustness is dramatically improved with the use of more data.

4. CONCLUSIONS

In this paper, we have presented four algorithms for automatically building example-tracing tutor models using multiple solution demonstrations that may be crowd-sourced or collected from a sample of users in an online ITS. The transfer of this effort from the ITS developers to a low cost (potentially no cost) workforce affords scale to the ITS development process.

Foremost, we must note that due to the inaccuracies in the automatically generated behavior graphs, they need manual inspection and further annotation before they can be operationalized. In our work on creating a general purpose learning platform focused on STEM domains, we are integrating these algorithms into our suite of authoring tools to allow ITS developers to use these algorithms in their workflow. Second, we notice that the algorithms have complementary performance on the different desirable characteristics of the automatically generated behavior graphs. Based on Table 2, we would choose Algorithm 2 for its Readability metrics, Algorithm 4 for Accuracy, Algorithm 1 for Completeness and Algorithm 3 for the key Robustness metric. All of these algorithms should be made available to the ITS developers through the authoring tools. We think that Algorithm 2 may be used as the default choice.

Looking ahead, the pursuit of automation of example tracing tutor modeling has a number of challenges of interest. The complementary nature of these algorithms suggests the potential for combining them to obtain better behavior graphs. Extension of the techniques presented in this paper to automatically update existing behavior graphs, which may have been manually authored, using traces from actual learners can help in maintenance and online improvement of the tutor models.

5. REFERENCES

[1] Aleven, V., Mclaren, B. M., Sewall, J., and Koedinger. K. R. 2009. A New Paradigm for Intelligent Tutoring Systems: Example-Tracing Tutors. *Int. J. Artif. Intell. Ed.* 19, 2 (April 2009), 105-154.

[2] Kumar, R., Roy, M.E, Roberts, R.B., and Makhoul, J.I. 2014. Towards Automatically Building Tutor Models Using Multiple Behavior Demonstrations. *12th Intl. Conf. on Intelligent Tutoring Systems* (ITS 2014), Honolulu, HI.

[3] Barnes, T., and Stamper, J. 2008. Toward Automatic Hint Generation for Logic Proof Tutoring Using Historical Student Data. *9th International Conference on Intelligent Tutoring Systems* (ITS 2008). Montreal, Canada.

[4] Eagle, M., Johnson, J., and Barnes, T., 2012. Interaction Networks: Generating High Level Hints Based on Network Community Clusterings, *5th International Conference on Educational Data Mining* (EDM 2012). Chania, Greece.

[5] McLaren, B.M., Koedinger, K.R., Schneider, M., Harrer, A., and Bollen, L. 2004. Bootstrapping Novice Data: Semi-Automated Tutor Authoring Using Student Log Files. *Workshop on Analyzing Student-Tutor Interaction Logs to Improve Educational Outcomes, 7th International Conference on Intelligent Tutoring Systems* (ITS 2004). Alagoas, Brazil.

[6] Johnson, M., Eagle, M., Stamper, J., and Barnes, T. 2013. An Algorithm for Reducing the Complexity of Interaction Networks, *6thInternational Conference on Educational Data Mining* (EDM 2013). Memphis, TN.

[7] Koedinger, K.R., Baker, R.S.J.d., Cunningham, K., Skogsholm, A., Leber, B., and Stamper, J. 2010. A Data Repository for the EDM community: The PSLC DataShop. In *Handbook of Educational Data Mining*. Romero, C., Ventura, S., Pechenizkiy, M., Baker, R.S.J.d. (Eds.). Boca Raton, FL: CRC Press