The Impact of Data Quantity and Source on the Quality of Data-Driven Hints for Programming

Thomas W. Price(✉), Rui Zhi, Yihuan Dong, Nicholas Lytle, and Tiffany Barnes

North Carolina State University, Raleigh, NC 27606, USA
{twprice,rzhi,ydong2,nalytle,tmbarnes}@ncsu.edu

Abstract. In the domain of programming, intelligent tutoring systems increasingly employ data-driven methods to automate hint generation. Evaluations of these systems have largely focused on whether they can reliably provide hints for most students, and how much data is needed to do so, rather than how useful the resulting hints are to students. We present a method for evaluating the quality of data-driven hints and how their quality is impacted by the data used to generate them. Using two datasets, we investigate how the quantity of data and the source of data (whether it comes from students or experts) impact one hint generation algorithm. We find that with student training data, hint quality stops improving after 15–20 training solutions and can decrease with additional data. We also find that student data outperforms a single expert solution but that a comprehensive set of expert solutions generally performs best.

Keywords: Data-driven hints · Programming · Hint quality Cold start

1 Introduction

Intelligent tutoring systems (ITSs) increasingly use student data to drive their decision making. Rather than relying on extensive knowledge engineering, authors can employ data-driven methods to automate the development of both "outer loop" [27] components of an ITS (e.g. constructing [30] and improving [10] student models) and "inner loop" components (e.g. automatically generating hints [1] and worked examples [14]). Authors of data-driven systems argue that these approaches avoid the need for experts to spend time constructing complex domain models [1,23] and can lead to additional insights that experts alone would not achieve [10].

However, empirical evaluations of the costs and benefits of data-driven approaches are still rare. This is especially true in the case of data-driven hint generation, where student data is used to automatically generate hints, rather than relying on expert models (e.g. model tracing [3] or constraint-based modeling [13]). Many evaluations of data-driven hints have focused on whether a

C. Penstein Rosé et al. (Eds.): AIED 2018, LNAI 10947, pp. 476–490, 2018.
https://doi.org/10.1007/978-3-319-93843-1_35

system can reliably provide hints to students [16,28], as well as how much data is necessary to do so [1,15,23], a challenge called the "cold start problem" [1]. However, a cold start analysis can only measure the availability of hints; it makes no claims about how useful the hints will be for students. This is particularly concerning in the domain of programming, where a recent evaluation of data-driven hints suggests that their quality varies considerably and that lower quality hints can deter students from seeking help when they need it [21].

In this paper, we present a reframing of the cold start problem in which we evaluate the *quality*, rather than the *availability*, of data-driven hints for programming. We use this analysis to investigate the following research questions:

RQ1: How does the quantity of available training data affect the quality of data-driven programming hints?

RQ2: How does data from students data compare to expert-authored data for generating high-quality programming hints?

We address these questions using datasets from two different programming environments that offer data-driven hints. We find that with student training data, hint quality stops improving after adding 15–20 training solutions and that additional data may harm hint quality. We also find that student data can outperform a single expert solution, but that a comprehensive set of expert solutions generally performs best. Our results suggest that some data-driven hint algorithms can be oversensitive to individual training solutions. We show that this can be reduced by weighting hints using multiple training solutions, which also significantly improves hint quality. The primary contributions of this work are a new procedure for assessing the quality of data-driven hints and results from the first investigation of the impact of data quantity and source on data-driven hint quality.

2 Related Work

The Hint Factory [1] was the first data-driven hint generation algorithm, originally used for logic proof problems. It constructs an Interaction Network [4] that models how students progress through discrete states during problem solving. When a student requests a hint, the Hint Factory looks up their current state in the network and, if it finds a match, suggests a successive state based on how other students successfully solved the problem. More recent approaches to data-driven hint generation have focused on the domain of computer programming. Because programming problems have a large space of possible states [19,23], it is often infeasible to match hint-requesting students exactly to other students in the database, as the Hint Factory does. Researchers have addressed this by finding the closest match, rather than an exact one [18,23,31], matching only some parts of the program [11,19,28], matching the student to a cluster of solutions [6], or matching using program output instead of source code [15].

2.1 The Cold Start Problem

Barnes and Stamper [1] recognized that one challenge with the Hint Factory is that when it does not have enough data, it will be unable to provide hints to students who have no match in the network – the "cold start problem." They devised a procedure to investigate how much data would be needed to reliably provide hints to students. They added student attempts to the network, one at a time, and after each addition calculated the likelihood that the next student would have a hint available, averaged over many trails. This produced a "cold start curve," which plots the amount of training data used to generate hints against the percent of students with a hint available. Their results showed that 75% hint coverage can be achieved with as few as 49 training attempts, improving to 91% coverage with over 500 attempts. A later evaluation [25] showed that with a few expert-authored "seed" solutions in the training dataset, hint coverage starts at over 55% and then quickly reverts to the original data-only curve.

Others have performed similar cold start experiments to show that data-driven hint generation for programming requires relatively little student data to achieve high hint coverage [2,5]. Peddycord et al. [15] used cold start curves to show how a different representation of student state improved hint availability. Rivers and Koedinger's ITAP tutor [23] can generate hints for over 98% of student attempts, since ITAP matches students with the closest partial solution in its dataset. Rather than measuring *hint availability* in their cold start analysis, they measured path length, or the number of edits needed to fully correct a student's code, which generally decreased as training data increased.

2.2 Other Evaluations of Data-Driven Hints

Other methods for evaluating data-driven hints include the following categories:

Availability Evaluations estimate how often a system will be able to provide a hint to a requesting student using all training data. Perelman et al. [16] found that their test-driven synthesis algorithm could generate hints for 65% of incorrect attempts in the CodeHunt programming game, while the iGrader system [28] could generate hints for 54% of incorrect attempts in a C# course.

Correction Evaluations estimate how often a system can *correct* an incorrect problem attempt. For the attempts where iGrader [28] was able to generate hints, it could fully correct 72% of these attempts with at most three "fixes." Lazar and Bratko's hint system [11] was able to correct 35–70% of incorrect attempts at a set of Prolog problems. The DeepFix system [7] could correct 27% of incorrect C programs fully and an additional 19% partially.

Student Choice Evaluations generate hints for students seen in historical data and then determine whether the students' later edits align with the hints' suggestions. Lazar et al. [12] found that 97% and 56% of their "buggy" and "intent" hints, respectively, were later enacted by students. Price et al. [19] found that 32–35% of their CTD algorithm's hints would have brought students closer to their own submitted solution, while 61% of students' actions did so.

Student Outcome Evaluations investigate the impact of hints on student outcomes in classroom and laboratory studies. Fossati et al. [5] found that their iList tutor, which provides data-driven hints, led to learning gains comparable to working with a human tutor. Stamper et al. [26] compared a version of their DeepThought logic tutor with and without data-driven hints across semesters and found that students with hints performed better in the tutor and the course.

Expert Evaluations directly measure the quality of data-driven hints using experts. Stamper and Barnes [24] compared two hint generation methods by having an expert select which method generated the better hint. Others have asked experts to rate hints directly on a validity scale [8,21,29]. Piech et al. [17] asked a group of experts to generate single, "gold standard" hints for student programs, and they evaluated hint systems based on their accuracy in matching these gold standard hints. Price et al. [18] extended this approach by having experts identify a *set* of valid hints, rather than a single best hint. Gross et al. [6] found that experts often disagreed with their cluster-based hint generation system about which exemplar solution should be used to generate feedback.

We argue that the first three evaluation categories are insufficient to demonstrate that data-driven hints are useful to students, since even hints that lead to a correct solution may not be easily understood [20]. While student outcome evaluations are ideal for evaluating whole systems, in this paper we focus on expert evaluation to understand how data impacts hint quality.

3 Method

We used the SourceCheck data-driven hint generation algorithm [18] to investigate our research questions. Like many hint generation algorithms in the domain of programming (e.g. [23,28,31]), SourceCheck generates hints for a hint-requesting student in two phases. First, it searches a training dataset of correct solutions for the one that best matches the hint-requesting student's code, using a code-specific distance metric. It then constructs a set of edits to transform the student's code into the matching solution and suggests these edits as hints.

3.1 Data

We analyzed datasets from two programming environments that offer on-demand, data-driven hints: iSnap [20], a block-based programming environment and ITAP [23], an intelligent tutoring system (ITS) for Python programming. Both datasets consist of log data collected from students working on multiple programming problems, including complete traces of their code and records of hints they requested. The iSnap dataset was collected from an introductory programming course for non-majors during the Fall 2016 and Spring 2017 semesters, with 120 total students completing 6 programming problems. The ITAP dataset was collected from two introductory programming courses in Spring 2016, with 89 total students completing up to 40 Python problems (see [22] for details). Both datasets are available from the PSLC Datashop (pslcdatashop.org) [9].

Our evaluation of data-driven hint quality required two sets of data: a set of *hint requests* for which to generate hints, and a set of *training data* used to generate those hints. From the iSnap dataset, we randomly sampled one hint request per problem from each student who used hints. This ensured that no student was overrepresented in the set of hint requests. From the ITAP dataset, we randomly sampled up to two unique hint requests per problem from each student who used hints, since there were fewer students who requested hints than in the iSnap dataset. We only sampled hint requests where the student's Python code could be parsed, since this is required by SourceCheck. We also extracted a set of training data from each dataset consisting of the traces of each student who submitted a correct solution and did not request hints.

We selected a subset of problems from each dataset to analyze, since our method for evaluating hint quality (explained in Sect. 3.2) involves time-intensive hint generation by expert tutors. From the iSnap dataset, we selected two representative problems, GuessingGame and Squiral, which have been used in previous evaluation of SourceCheck [18]. The two problems had 23 and 24 hint requests respectively (47 total). Common solutions to these problems are approximately 13 and 10 lines of code, respectively, and require loops, conditionals, variables and procedure definitions. From the ITAP dataset, we selected all 5 problems that had at least 7 associated hint requests, for a total of 51 hint requests (7–14 per problem). These simpler problems had common solutions with 2–5 lines of code, which included variables, API calls, arithmetic operators and, for one problem, loops. An important difference between the datasets is that the iSnap problems are considerably longer and more open-ended, while the ITAP problems often involve single-line functions, evaluated with test cases.

3.2 Assessing Hint Quality

To address our RQs, we developed a method to measure data-driven hint quality. As in previous work [17,18], we used a group of experts to generate a set of "gold standard" hints for each hint request, and we labeled data-driven hints as high-quality if they matched the gold standard hints. First, three tutors independently reviewed each hint request, including the history of the student's code before the hint request. Each tutor then generated a set of all hints they considered to be valid, useful, and not confusing. Each hint was represented as one or more edits to the student's code, making these hints comparable to the edit-based hints offered by many data-driven algorithms. Tutors were instructed to limit their hints to one edit (e.g. one insertion) unless multiple edits were needed to avoid confusion. Hints were designed to be independently useful, with the understanding that the student would receive any *one* hint, not the whole set.

Next, each tutor independently reviewed the hints generated by the other two tutors and marked each hint as valid or invalid by the same criteria used to generate hints. We included in our gold standard set any hint which at least two out of three tutors considered to be valid. Our goal was not to determine a definitive, objective set of correct hints but rather to identify a large number of hints that a reasonable human tutor might generate. Requiring that two tutors agreed on each hint provided a higher quality standard than is used in most classrooms, while allowing for differences of opinion among tutors. This produced between 1 and 11 gold standard hints per hint request for the iSnap dataset (Med = 5) and between 1 and 5 for the ITAP dataset (Med = 2)[1].

We use this set of gold standard hints to automatically assign a QualityScore to a hint generation algorithm for a set of hint requests. For each hint request, we first generate a set of candidate hints, H, using the algorithm. Since some algorithms (including SourceCheck) generate multiple hints for a given request, the algorithm must also assign a confidence weight to each hint, with the weights summing to 1. We then mark each candidate hint $h \in H$ as valid if it matches one of the gold standard hints, after standardizing the names of any newly created variables, methods and string literals. We also detect *partial* matches where a candidate hint consists of a subset of the edits suggested by the tutor. The final QualityScore is the sum of the weights of all valid hints in H. The QualityScore ranges from 0 (no valid hints) to 1 (all valid hints) and can be calculated to include partial matches or only full matches. We average this score over all hint requests to produce a final QualityScore for the algorithm.

In our analysis, we compared two different approaches for assigning weights to the set of hints generated by SourceCheck, H, for a given hint request. The first, uniform weighting, simply assigns a weight of $1/|H|$ to each hint. The second, voting-based weighting, uses multiple solutions in the training dataset to assign weights. Recall that SourceCheck uses a single, closest matching target solution in the training dataset to generate hints (a 1-nearest-neighbor approach). With voting-based weighting, hints are also generated using the top-k closest solutions in the training dataset. The weight of each hint $h \in H$ is the percentage of these top-k solutions which, if used for hint generation, would also have generated h. The weights are then normalized to sum to 1. In our analysis we chose $k = 10$, though other values of $k = 5 \ldots 15$ produced similar outcomes. Note that the weighting approach does not change which hints are generated, only each hint's weight in determining the QualityScore.

3.3 Cold Start Procedure

To investigate RQ1, we developed a cold start procedure that measures the relationship between the quantity of available data and the resulting hint QualityScore. Unlike other cold start procedures (e.g. [1,5,23]), our evaluation considers only *actual* student hint requests, rather than all states observed in historical data. For each dataset, we start with the full set of training traces, T, the set of hint requests, R, and a sample set of training traces, $S = \emptyset$. We repeatedly select a random trace in T, remove it and add it to S. After each addition, we use the SourceCheck algorithm to generate a set of hints for each hint request in R and calculate a QualityScore for these hints. We repeat this until T is empty. The end result is a QualityScore for each $i \in 1..|T|$,

[1] Full datasets and hint ratings are available at go.ncsu.edu/hint-quality-data.

where i is the number of traces used to generate hints. We repeated this whole procedure for the iSnap and ITAP datasets 200 times and averaged the results[2].

3.4 Comparing with Expert-Authored Data

In order to investigate RQ2, comparing data from students and expert-authored data, we created two expert-authored training datasets to use as baselines. The first baseline dataset, OneExpert, consists of a single solution, authored by an expert to reflect a straightforward approach to solving each problem. This is a minimal baseline, since almost any problem will already have at least one teacher-authored solution available. The second baseline dataset, AllExpert, consists of *all* solutions considered by an expert to be useful for hint generation. To assign a QualityScore to these baselines, we still generate hints with the SourceCheck algorithm, but we provide it the expert training dataset.

To allow an expert author to generate a comprehensive set of correct solutions, we developed a simple solution templating language. Using this language, an author can write solution code that includes "branches," indicating where there are multiple ways of writing an acceptable solution. These branches may represent different high-level programming strategies, or different ways of expressing the same idea programmatically. Branches can be nested to produce a wide variety of solutions. Additionally, authors can mark parts of the solution code as order-invariant and add wildcards that can match any code element. One researcher, familiar with the problems, authored the expert solutions for both baselines, using the set of students solutions as a reference. For the larger iSnap problems, AllExpert solution sets took 60–90 min each to author and test, producing 960 and 1,152 unique solutions for GuessingGame and Squiral, respectively. The smaller ITAP problems took 15–60 min each, producing only 2 solutions for most problems, though one problem (KthDigit) had 16.

4 Results

Figure 1 shows cold start curves for the iSnap dataset, plotting how SourceCheck's QualityScore (averaged over all trials) changes for both weighting approaches as the number of solutions in the training dataset increases. In many ways, the curves resemble a traditional cold start curve, with the QualityScore increasing rapidly as the first 10–15 solutions are added to the training data and then leveling off. Comparing the uniform and voting-based weighting approaches, we can see that the voting-based approach consistently performs better, achieving a 19.1% and 15.4% higher final QualityScore on the GuessingGame and Squiral problems, respectively. A Wilcoxon signed-rank test showed that this difference in the final QualityScore was significant for the sample of 47 iSnap hint requests ($V = 331$; $p = 0.015$; Cohen's $d = 0.143$).

[2] Over 200 trials, the standard error of the averaged QualityScores was always less than 0.01, and averaged less than 0.0025 across values of i.

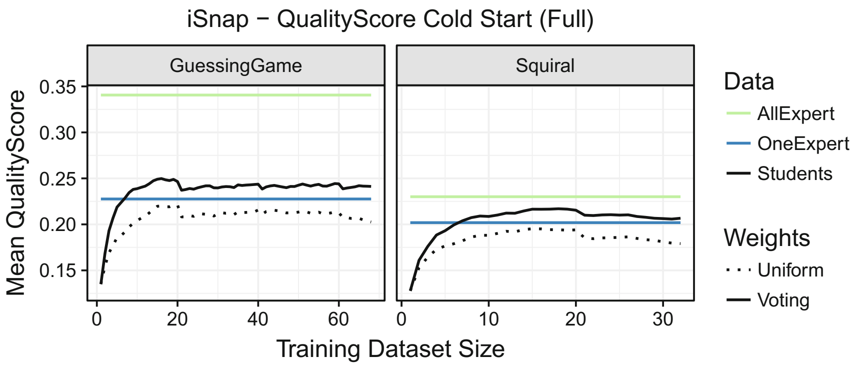

Fig. 1. Cold start curves for the iSnap dataset (black) with uniform (solid) and voting-based weighting (dotted) with AllExpert (green) and OneExpert (blue) baselines. (Color figure online)

Traditional cold start curves measure hint *coverage* and are guaranteed to increase monotonically with more data. However, our QualityScore curves show clear fluctuations as more training solutions are added, sometimes decreasing in QualityScore. Here too we see a difference between weighting approaches. The uniform weighting curves have a final QualityScore 8.4% and 8.3% lower than their peak score on GuessingGame and Squiral, respectively, while the voting-based weighting curves lose only 3.4% and 4.8% of their peak QualityScore, respectively. Recall that Fig. 1 shows QualityScore averaged over all hint requests for a given problem. To further investigate why additional data may decrease hint QualityScore in the iSnap dataset, we calculated cold start curves for each of the 47 iSnap hint requests individually, and we found a wide variety of curve shapes. For voting-based weighting, 39.1% and 33.3% of hint requests on GuessingGame and Squiral showed a negative correlation between quantity of data and hint QualityScore, while 47.8% and 45.8% showed a positive correlation. This trend was similar for uniform weighting. This suggests the fluctuations at the end of the cold start curves are due to competing positive and negative effects of data quantity on different hint requests.

Figure 1 shows lines indicating the OneExpert and AllExpert baseline QualityScores[3]. They have a constant value, since they are calculated using a set of expert solutions, not student data. Student data with uniform weighting fails to surpass the OneExpert baseline. However, with voting-based weighting, student data outperforms OneExpert with only 7 training solutions for both problems, achieving a peak QualityScore 9.8% and 7.5% higher than OneExpert on GuessingGame and Squiral. Even with voting-based weighting, student data falls short of the AllExpert baseline, with the peak student

[3] Because the OneExpert baseline uses only one training solution, both weighting approaches produce the same results. For simplicity, Fig. 1 shows only voting-based weighting for the AllExpert baseline.

QualityScore 73.3% and 94.3% as high as the AllExpert baseline. However, a Wilcoxon signed-rank test shows that this difference between the final student data QualityScore and the AllExpert baseline was not significant for the sample of 47 iSnap hint requests ($V = 181$; $p = 0.077$; Cohen's $d = 0.221$).

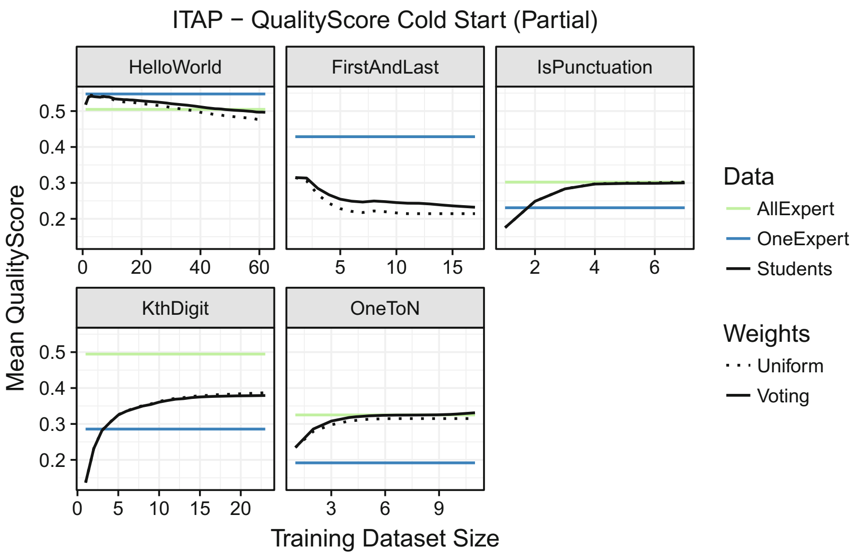

Fig. 2. Cold start curves and baselines for the ITAP dataset.

Figure 2 shows cold start curves for the ITAP dataset. Note that unlike in the iSnap dataset, QualityScores for ITAP curves were calculated including *partial* matches (see Sect. 3.2), since Python ASTs are more complex, and many of SourceCheck's hints matched only part of the gold standard hints. The IsPunctuation, KthDigit, and OneToN problems have a similar curve shape to the iSnap problems; however, rather than fluctuating, they increase monotonically in QualityScore with additional data. They also show almost no difference between uniform and voting-based weighting. Student data easily outperforms the OneExpert baseline and, in the IsPunctuation and OneToN problems, matches the AllExpert baseline. These problems have much smaller training datasets, so it is unclear how the curves would continue with additional training data.

The other two curves, for HelloWorld and FirstAndLast, have unique shapes. For both problems, additional data has a clear negative impact on hint quality. Additionally, the OneExpert baseline has equal or better performance compared to the AllExpert baseline. HelloWorld and FirstAndLast are the simplest problems we investigated, each with a common, one-line solution that comprised 47.1% and 93.5% of the training dataset, respectively. This solution was also used in the OneExpert baseline. The curves suggest that the presence of other solutions in the training dataset lowered hint QualityScore. For FirstAndLast, the second most common solution comprised 35.3% of the training dataset. It used a more advanced Python feature (using `s[-1]` to get the last character in a string), which the human tutors never suggested to struggling students, preferring a more straightforward approach (`s[len(s) - 1]`). SourceCheck often selected the more advanced approach as the target solution, since it is shorter, which accounts for the lower QualityScore for student data on this problem. It is also important to note that QualityScores for these two problems were based on the fewest number of hint requests (7 each), which may not be fully representative of all likely hint requests.

5 Discussion

RQ1: *How does the quantity of available training data affect the quality of data-driven programming hints?* Our results suggest increased training data has a positive impact on hint quality up to a point. For our datasets, hint quality stops increasing after 15–20 training solutions. However, for the more complex problems in the iSnap dataset, additional data had either no positive effect, or even a slight negative effect, on hint quality. While this does validate the claim of many data-driven algorithms that little student data is needed to generate hints [1,5], it is concerning that additional data fails to improve SourceCheck's effectiveness. A similar problem was noted by Rivers et al. [23] on some problems, where ITAP's hint generation performance decreased with additional data. We could interpret the analysis presented here as a method to measure a data-driven algorithm's ability to effectively make use of all of its data. Ideally, a data-driven algorithm should have monotonically increasing hint quality as it is given additional training data, selecting useful solutions and ignoring unhelpful ones. By this interpretation, the SourceCheck algorithm has mixed success, especially on the iSnap dataset.

In considering why additional data may be harmful, recall that this evaluation relied on the SourceCheck algorithm to generate all the hints we evaluated. Like many data-driven hint generation algorithms (e.g. [23,28,31]), SourceCheck generates hints using the single best-matching target solution from the training dataset (a "1-nearest neighbor" approach). This makes it sensitive to additional training data. Our training data consists of *all correct* student solutions, and some of these will contain unnecessary code, unique design choices, or advanced features (e.g. `s[-1]` in FirstAndLast) that make them problematic for hint generation. This is especially true for the more open-ended problems found in the iSnap dataset [20]. As more of these problematic solutions are added to the training dataset, it becomes increasingly likely that one of them will be the closest match for the hint-requesting student (e.g. if the student added similar unnecessary code). Since only the closest match in the training dataset is used for hint generation, a single problematic solution can eclipse others that are more useful.

Our comparison of uniform and voting-based hint weighting lends some support to the hypothesis that this "1-nearest-neighbor" approach to hint generation is overly sensitive to additional data. When SourceCheck uses the top 10 best matches in the training dataset to weight hints, its final QualityScore improves significantly and less quality is lost from additional training data. However, this weighting does not change the hints themselves, which may still suffer from a bad match. Another way to reduce this sensitivity to additional data is the approach of Gross et al. [6], which selects a matching *cluster* of solutions, rather than a single target, and uses an exemplar solution from that cluster for hint generation. Rivers and Koedinger [23] address this in part by discovering new target solutions through path construction, which better match the hint requesting student's code. We could also attempt to filter training solutions (manually or automatically) to retain only those useful for hint generation. Importantly, both the decrease in QualityScore with additional training data and the difference in QualityScore between weighting approaches appeared only in the iSnap dataset. This may be because the iSnap dataset featured larger and more complex problems than the ITAP dataset, but it is also possible that these trends result from more specific features of the iSnap dataset itself.

RQ2: *How does data from students compare to expert-authored data for generating high-quality programming hints?* The use of student data for hint generation makes the implicit assumption that it will yield better hints than using a single expert solution, and our results suggest that this is not always the case. In the iSnap dataset, student data failed to outperform the OneExpert baseline when using uniform hint weighting. However, we also showed that this can be addressed with voting-based weighting. On the simpler ITAP dataset, student data seems to perform better, outperforming OneExpert with only 1–3 training solutions on 3 problems, regardless of the hint weighting. However, our results also show that for the simplest problems (HelloWorld and FirstAndLast), a single expert solution may be all the data that is required.

The AllExpert baseline is more robust, and it outperformed student data on the three problems where more than 2 expert solutions were generated as training data (GuessingGame, Squiral, KthDigit), as well as on FirstAndLast. While this difference was not significant on the iSnap dataset, this may be due to our relatively small sample size (47 hint requests). On the remaining problems, student data has similar performance to the AllExpert baseline. These results suggest that for all but the simplest problems, a comprehensive set of expert solutions is likely to be more useful for hint generation than student data. This is not surprising, since our expert solutions were authored by a researcher familiar with the problems (as most instructors would be), using student solutions as a reference, so in many ways they represent an ideal training dataset. It would also be possible to author expert solutions without the benefit of referencing student solutions, but this may yield lower-quality hints.

The template-based data generation technique used in the AllExpert baseline is not the focus of this paper, but we do note some advantages to this approach. In addition to outperforming student data in terms of QualityScore, this approach solves the cold start problem entirely. Also, if a problem changes from one semester to the next (e.g. adding an additional objective), it is easy to modify the corresponding template, while any student solutions would be rendered useless. Previous work suggests that even a few low-quality hints may deter students from using a help system [21], so the consistency of expert-authored data is desirable. However, like expert models, the templates are time intensive to produce, taking 15–90 min per problem and likely longer for more complex problems. This process may therefore not scale well to larger problem sets.

5.1 Considerations and Limitations

Automatically assessing the quality of hints is a difficult and inherently subjective task. Our method for generating gold standard tutor hints strikes a balance between including many hints (generated by three separate tutors) and ensuring hint quality (all hints were "seconded" by another tutor). However, our results may have been different if we had used more than 3 tutors or a higher standard of agreement (consensus among all tutors). The process of matching an automatically-generated hint to a gold standard hint is also imperfect, and it is likely that some hints were marked invalid despite being similar to a gold standard hint. For all of these reasons, the QualityScores reported here may seem low. However, raw QualityScores are difficult to interpret, so we use them primarily for *comparing* hint generation approaches.

One important limitation of this work is that the QualityScore metric was calculated based on a moderate number of hint requests for each problem (23–24 for iSnap; 7–14 for ITAP). The QualityScore metric is therefore only as accurate as these hint requests are representative of all hint requests. The tutors generating hints for these requests noted that students' code often contained a range of strange misconceptions and creative mistakes and often did not resemble other students' solutions. However, we see this as an important feature of our data and analysis. The set of code states where students request hints is *not* representative of students' code generally, since it reflects students' need to request outside help. Hint requests are full of strange design choices and incorrect code, and this is exactly what a hint generation algorithm should be designed to address. One limitation of this work is that these hint-requesting students were not included in the training datasets, diminishing their diversity. Lastly, most problems in the ITAP dataset had fewer training solutions (7 to 22), and it is unclear how the cold start curves would have continued with additional data.

6 Conclusion

In this paper we presented a detailed method for evaluating the quality of data-driven hints, and we applied that method to investigate how the quantity and source of data impacts data-driven hint quality for one algorithm. We showed that a relatively small number of student solutions can outperform a single expert-authored solution, but they fall short of a comprehensive set of expert-authored solutions. We also introduced a voting-based hint weighting approach that significantly improves hint quality. We compared our results across 2 datasets that used different programming languages – to our knowledge, the first evaluation of data-driven hints to do so. While our quality evaluation procedure comes with limitations, we argue that research on data-driven hint generation should hold itself to the more rigorous standard of evaluation presented here, rather than relying on availability evaluations. We have suggested that part of that standard should be an algorithm's ability to effectively leverage training data, as assessed with the cold start curves presented here. In future work, we hope to use our method to benchmark and compare hint generation algorithms on common datasets to better understand their strengths and weaknesses.

Acknowledgements. The authors thank Veronica Cateté for her contributions to this research. This work was supported by the National Science Foundation under grant 1623470.

References

1. Barnes, T., Stamper, J.: Automatic hint generation for logic proof tutoring using historical data. J. Educ. Technol. Soc. **13**(1), 3 (2010)
2. Chow, S., Yacef, K., Koprinska, I., Curran, J.: Automated data-driven hints for computer programming students. In: Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization, pp. 5–10 (2017)
3. Corbett, A.T., Anderson, J.R.: Locus of feedback control in computer-based tutoring: impact on learning rate, achievement and attitudes. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 245–252 (2001)
4. Eagle, M., Johnson, M., Barnes, T.: Interaction networks: generating high level hints based on network community clustering. In: International Educational Data Mining Society (2012)
5. Fossati, D., Di Eugenio, B., Ohlsson, S., Brown, C., Chen, L.: Data driven automatic feedback generation in the ilist intelligent tutoring system. Technol. Inst. Cogn. Learn. **10**(1), 5–26 (2015)
6. Gross, S., Mokbel, B., Paaßen, B., Hammer, B., Pinkwart, N.: Example-based feedback provision using structured solution spaces. Int. J. Learn. Technol. 10 **9**(3), 248–280 (2014)
7. Gupta, R., Pal, S., Kanade, A., Shevade, S.: Deepfix: fixing common C language errors by deep learning. In: AAAI, pp. 1345–1351 (2017)
8. Hartmann, B., MacDougall, D., Brandt, J., Klemmer, S.R.: What would other programmers do: suggesting solutions to error messages. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1019–1028 (2010)
9. Koedinger, K.R., Baker, R.S., Cunningham, K., Skogsholm, A., Leber, B., Stamper, J.: A data repository for the EDM community: the PSLC DataShop. In: Handbook of Educational Data Mining, p. 43 (2010)

10. Koedinger, K.R., Stamper, J.C., McLaughlin, E.A., Nixon, T.: Using data-driven discovery of better student models to improve student learning. In: Lane, H.C., Yacef, K., Mostow, J., Pavlik, P. (eds.) AIED 2013. LNCS (LNAI), vol. 7926, pp. 421–430. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39112-5_43
11. Lazar, T., Bratko, I.: Data-driven program synthesis for hint generation in programming tutors. In: Trausan-Matu, S., Boyer, K.E., Crosby, M., Panourgia, K. (eds.) ITS 2014. LNCS, vol. 8474, pp. 306–311. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07221-0_38
12. Lazar, T., Možina, M., Bratko, I.: Automatic extraction of AST patterns for debugging student programs. In: André, E., Baker, R., Hu, X., Rodrigo, M.M.T., du Boulay, B. (eds.) AIED 2017. LNCS (LNAI), vol. 10331, pp. 162–174. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61425-0_14
13. Mitrovic, A.: A knowledge-based teaching system for SQL. In: Proceedings of ED-MEDIA, vol. 98, 1027–1032 (1998)
14. Mostafavi, B., Zhou, G., Lynch, C., Chi, M., Barnes, T.: Data-driven worked examples improve retention and completion in a logic tutor. In: Conati, C., Heffernan, N., Mitrovic, A., Verdejo, M.F. (eds.) AIED 2015. LNCS (LNAI), vol. 9112, pp. 726–729. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19773-9_102
15. Peddycord III, B., Hicks, A., Barnes, T.: Generating hints for programming problems using intermediate output. In: Proceedings of Educational Data Mining (2014)
16. Perelman, D., Gulwani, S., Grossman, D.: Test-driven synthesis for automated feedback for introductory computer science assignments. In: Proceedings of Data Mining for Educational Assessment and Feedback (ASSESS 2014) (2014)
17. Piech, C., Sahami, M., Huang, J., Guibas, L.: Autonomously generating hints by inferring problem solving policies. In: Proceedings of the Second ACM Conference on Learning@ Scale, pp. 195–204 (2015)
18. Price, T., Zhi, R., Barnes, T.: Evaluation of a data-driven feedback algorithm for open-ended programming. In: Proceedings of Educational Data Mining (2017)
19. Price, T.W., Dong, Y., Barnes, T.: Generating data-driven hints for open-ended programming. In: Proceedings of Educational Data Mining, pp. 191–198 (2016)
20. Price, T.W., Dong, Y., Lipovac, D.: iSnap: towards intelligent tutoring in novice programming environments. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, pp. 483–488 (2017)
21. Price, T.W., Zhi, R., Barnes, T.: Hint generation under uncertainty: the effect of hint quality on help-seeking behavior. In: André, E., Baker, R., Hu, X., Rodrigo, M.M.T., du Boulay, B. (eds.) AIED 2017. LNCS (LNAI), vol. 10331, pp. 311–322. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61425-0_26
22. Rivers, K., Harpstead, E., Koedinger, K.R.: Learning curve analysis for programming: which concepts do students struggle with? In: Proceedings of the 12th Annual International ACM Conference on International Computing Education Research, pp. 143–151 (2016)
23. Rivers, K., Koedinger, K.R.: Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. Int. J. Artif. Intell. Educ. **27**(1), 37–64 (2017)
24. Stamper, J., Barnes, T.: An unsupervised, frequency-based metric for selecting hints in an MDP-based tutor. In: Proceedings of Educational Data Mining (2009)
25. Stamper, J., Barnes, T., Croy, M.: Enhancing the automatic generation of hints with expert seeding. Int. J. Artif. Intell. Educ. **21**(1–2), 153–167 (2011)
26. Stamper, J., Eagle, M., Barnes, T., Croy, M.: Experimental evaluation of automatic hint generation for a logic tutor. Int. J. Artif. Intell. Educ. **22**(1–2), 3–17 (2013)

27. VanLehn, K.: The behavior of tutoring systems. Int. J. Artif. Intell. Educ. **16**(3), 227–265 (2006)
28. Wang, K., Lin, B., Rettig, B., Pardi, P., Singh, R.: Data-driven feedback generator for online programing courses. In: Proceedings of the Fourth ACM Conference on Learning@ Scale, pp. 257–260 (2017)
29. Watson, C., Li, F.W.B., Godwin, J.L.: BlueFix: using crowd-sourced feedback to support programming students in error diagnosis and repair. In: Popescu, E., Li, Q., Klamma, R., Leung, H., Specht, M. (eds.) ICWL 2012. LNCS, vol. 7558, pp. 228–239. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33642-3_25
30. Yudelson, M., Hosseini, R., Vihavainen, A., Brusilovsky, P.: Investigating automated student modeling in a Java MOOC. In: Proceedings of Educational Data Mining, pp. 261–264 (2014)
31. Zimmerman, K., Rupakheti, C.R.: An automated framework for recommending program elements to novices (n). In: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 283–288 (2015)