International Journal of Engineering Education Vol. 35, No. 1(A), pp. 252–261, 2019
Printed in Great Britain

0949-149X/91 $3.00+0.00

An Intelligent Tutoring System for Computer Numerical Control Programming*

QINBO LI[1] and SHENG-JEN HSIEH[2]
[1] Department of Computer Science & Engineering, Texas A&M University, USA.
[2] Department of Engineering Technology & Industrial Distribution and Department of Mechanical Engineering, Texas A&M University, College Station, TX 77843-3367, USA. E-mail: hsieh@tamu.edu

G-code is the language used to control computer numerical control (CNC) machines. Although most CAD/CAM software can generate G-code based on a design and machine tools needed, the ability to understand G-code is valuable, especially when a machining job does not run smoothly. Intelligent tutoring systems (ITS) have been shown to be successful in helping students to learn programming. However, G-code is different from general purpose computer programming languages. CNC programming also requires that students master many hardware-related facts and concepts, such as cutting speed and feed rate, and tools for several types of drilling. We built a web-based ITS for CNC programming called CNC-Tutor, and proposed a data-driven approach to generate proper hints and feedback during students' problem-solving process. The approach is based on finding past submissions that are most similar to a student's solution. The similarity is measured using a "behavior & machine state distance" metric. The system was evaluated by 93 undergraduate students. Results suggest that the design is instructionally effective, and that students' subjective impressions of the system were positive. It also appears that CNC Tutor's explanations and feedback are a good fit for active, visual learners.

Keywords: intelligent tutoring system; computer integrated manufacturing; mechanical engineering

1. Introduction

Intelligent tutoring systems (ITS) are computer-based teaching environments to help students learn deep domain knowledge [1, 2]. ITS may incorporate mathematics, cognitive science, natural language processing, and human-computer interaction [3]. In recent years, the use of ITS in classrooms and communities has increased and they have been shown to be effective [20, 21, 24–27]. However, to the best of our knowledge, an ITS for Computer Numerical Control does not currently exist.

Hsieh and Cheng [4] proposed a system architecture to help students learn ladder logic programming for programmable logic controllers (PLC). This system is a web-based tutoring system that contains pre-test, case-based reasoning, and post-test. It can also customize the learning sequence based on the students' knowledge levels, which are measured by students' feedback and user questionnaires.

BITS [5] is a Bayesian intelligent tutoring system for computer programming. It uses a Bayesian network to represent the structure of the problem as well as the students' knowledge. A Bayesian network consists of a directed acyclic graph (DAG) and a conditional probability distribution (CPD) table. Each node in the DAG is related to a concept in the domain. A directed link from a node to another node, say, node A to node B, indicates that node A is the prerequisite to learn node B. The DAG is constructed manually and the CPD table is obtained by previous exams in that course. The main functions that BITS provides are navigation support, prerequisite recommendations, and learning sequence generation. The navigation support function classifies each node into three categories: already known, ready to learn, and not ready to learn, based on the students feedback and the predefined CPD table. Next, the student can start to learn the recommended ready to learn nodes. Prerequisite recommendations provide recommended concepts when the student is stuck on the current node. The last function enables students to choose a particular lecture they want to learn without learning all the lectures before it. This is done by finding the minimum prerequisite based on the CPD table and the students' knowledge and then generating the learning sequence.

Another tutoring system for programming, QuizJET [6], supports authoring, delivery and assessment of parameterized questions for Java programming. Parameterized questions are patterns of questions that are created initially by domain experts, and the pattern is replaced by randomly generated parameters at the presentation time. This way, many similar questions can be used to evaluate students' performance. The questions cover critical topics in Java such as objects, classes, interfaces, inheritances, and exceptions. The advantage of this approach is that the authoring cost is significantly reduced and the possibility of plagiarism is also reduced. Students who solved more questions using this system showed better overall

* Accepted 1 October 2018.

performance in the final exam. QuizPACK (Quizzes for Parameterized Assessment of C Knowledge) [7], is a similar system for the C programming language.

JO-Tutor is another ITS for JAVA [8]. It can generate problems automatically based on randomly instantiated templates. The topics of the problem include functions, classes, inheritance, polymorphism, and so on. It also has an expert module to solve these problems to judge the response and provide feedback.

Automatically generating hints and feedback is another area of focus in programming tutors [9, 11, 22, 23]. Computer programming is known to be ill-defined [12] because the solution space is too large or even infinite, especially when incorrect solutions are taken into consideration. Rivers and Koedinger [13] proposed a data-driven approach to generate programming hints automatically. They use abstract syntax trees (ASTs) [10] to reduce the solution space: different programs with the same semantics can reduce to the same abstract syntax tree using copy propagation. The next step is to find the AST in prior student data that has the closest distance to the AST of the current program. The final step is generating hints based on the difference between the AST of the current program and the ASTs of a prior correct program.

Jin, et al. [9] proposed a hint generation approach that can generate hints automatically based on the completed solutions. They used a linkage graph to represent the solution space. In the linkage graph, the nodes are the program statements while the edges are the order dependencies. To generate hints, the algorithm first compares the linkage graph of a student's program with the linkage graph from the past-completed programs. It will then find the closest one in the Markov decision process (MDP) and generate hints based on the next best state in the MDP. Then the new linkage graph will be added to the dataset. Their experiment found that among 16 submissions, the approach was able to generate meaningful hints for 14 of them.

Table 1 shows a comparison of different ITS for programming. Note that "Feedback = Yes" means that detailed feedback information is provided, not just whether the solution is correct.

For the domain of computer numerical control, however, existing ITS development approaches are not applicable or will not work well. In addition learning problem solving skills, students must also master a large number of facts and concepts about the CNC machine before trying to write a program. Such concepts and facts include, for example, the physical characteristics of the machine, the usage of CNC programming terminology, the correct spindle, and feed rate [14, 15].

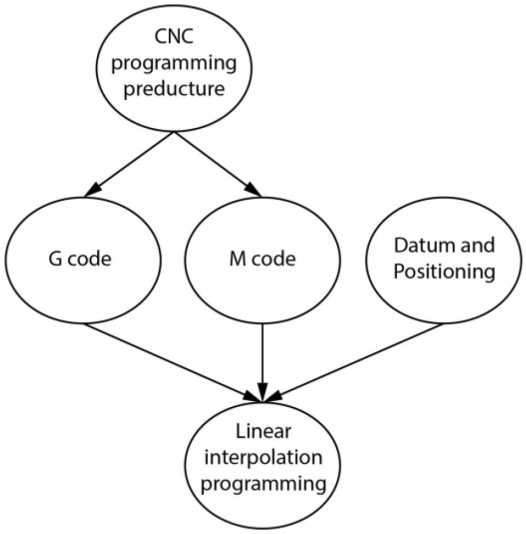

Fig. 1. Part of the DAG for the curriculum structure knowledge in the domain model.

We built an ITS for CNC called CNC-Tutor and proposed a data-driven approach that can generate proper feedback and hints during a student's problem-solving process. A Hint is a message of the next-step action that leads to the correct solution, and feedback is a list of messages about the missing facts or misconceptions that the student might have. This approach generates hints and feedback based on similar correct solutions from an archive of programs by former students. The similarity is measured by the proposed "Behavior & Machine state distance" metric. Experiments show that the generated hints can help the students solve the CNC problem and the generated feedback can help the students to find their misconceptions. A survey about the effectiveness of CNC-Tutor found a positive impact on the students.

Table 1. Comparison of different ITS for programming

	Target	Feedback	Hints	Web-based
BITS	C/C++	No	No	Yes
QuizJET	Java	No	0.008	Yes
QuizPACK	C++	No	No	Yes
JO-Tutor	Java	No	No	No
SQL-Tutor	SQL	Yes	No	Yes
PLC-Tutor	PLC	Yes	No	Yes

2. System architecture

In the domain of computer numerical control, there are two types of knowledge that students need to learn: concepts/facts and problem-solving skills. The CNC tutoring system consists of a learning module, a quiz module, and an exercise module. The learning module and the quiz module are focused on teaching concepts and facts, while the exercise module focuses on improving the students' problem-solving skills. Underlying these three modules are two data models: the domain model and the student model.

The domain model manages all the data related to the domain knowledge, such as what are the prerequisites of a lesson, how to generate questions for a lesson, what are the answers and feedback for a question, and so on. We use a directed acyclic graph (DAG) to represent the curriculum structure knowledge in the domain model. In the DAG, each node represents a piece of learning content. An edge from a source node to a destination node indicates that the source node is a prerequisite to learn the destination node. Currently, the system has 49 nodes and 65 edges. Fig. 1 shows a portion of the DAG.

The student model traces the students' state, for example, whether a student has passed a quiz, how many facts a student already knows, and so on. The student model is essentially a copy of the DAG with an additional attribute for each node, which is whether the student has mastered the learning content or not. There are many questions for each piece of learning content, and the system will randomly select one or more questions.

In addition, in order to evaluate students' CNC code and generate hints and feedback, there is a model called CNC interpreter under the exercise module. We noticed that students fail to solve a CNC programming problem usually because of the lack of problem solving skills as well as misconceptions or missing facts. CNC-Tutor can improve students' problem-solving skill by providing hints during their problem solving process. When the students submit an incomplete or incorrect solution, CNC-Tutor can generate feedback to help them figure the problem, such as information about the syntax error, and/or links to the learning content about their misconceptions. Fig. 2 illustrates the overall system architecture of CNC-Tutor.

3. Methodology

The key feature of CNC-Tutor is that it can generate proper hints or feedback as students are writing the CNC code. In the domain of ITS, the term "feedback" can represent many different kinds of messages provided by the system (examples, hints, correctness, etc.) [17]. However, in our system, a hint is a message of the next-step action that leads to the correct solution, while feedback is a list of messages about the missing facts or misconceptions that the students might have. Feedback is only provided when students fail to solve a problem.

Although CNC code is not as complex as general purpose programming code such as C++, the solution space of a CNC programming problem is still very large. First, there are many choices of cutting paths to cut a part, as long as the outcome matches with the blueprint. Even if the cutting paths are fixed, students can choose different settings to cut the part. In addition, CNC code can be written in

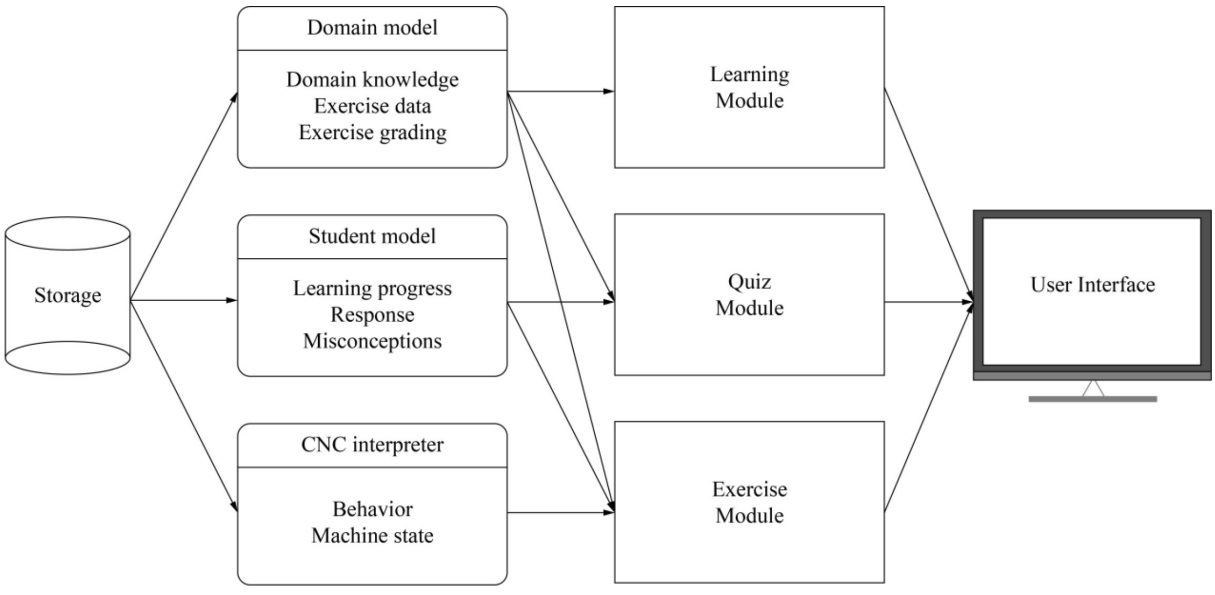

Fig. 2. System architecture of CNC-Tutor.

one line or in multiple lines with the same semantics. Third, there is no fixed order in which to write CNC code. Therefore, analyzing and generating hints for CNC code is challenging.

We noticed that if students follow the same path in the solution space (i.e., they choose the same strategy to solve a problem), the behavior (the cutting paths) of their programs will be the same, and the machine state's transition during the execution in the CNC simulator will be similar, regardless of individual differences in coding. We propose an approach that can compare the similarity between a current student's solutions and labeled past data to automatically generate proper feedback. The similarity of different CNC code is measured by using a novel construct that we call Behavior & Machine-state distance.

3.1 Behavior & machine-state distance

Behavior distance is computed by calculating the Euclidean distance between the cutting path of a current student's solution and past solutions. If the behavior distance is smaller than a specified tolerance, then those solutions follow the same cutting path. The behavior distance is calculated before the machine state distance, because it is meaningless to compare machine states if the cutting paths are different.

The machine state distance is calculated by comparing the difference of each state. Because the parameters, settings, and machine states are used to cause an effect on the parts, before comparing the distance between the machine states, their behavior must be the same. That is, the current coordination of the tool end and the motion to be executed must be the same. Therefore, each line of code in one program is aligned to the code that has the same behavior of another program. This process is called "Behavior alignment". Fig. 4 shows an example of "Behavior alignment".

Machine state is discussed in detail in the CNC interpreter section. For computational efficiency, only a portion of the machine state is recorded to calculate the machine state distance. Fig. 3 illustrates an example of CNC code and its corresponding machine states when the execution is at the line "N0090".

3.2 The CNC interpreter

To evaluate a CNC program and calculate Behavior & Machine-State Distance, we built a CNC interpreter. Because the CNC interpreter was developed in PHP and is heavily based on the National Institute of Standards and Technology (NIST) RS274NGC project [18], we named it RS274NGC-PHP.

The CNC program interpreter follows the RS274

```
N0010 G90 G20 G40 G80
N0020 G91 G28 X0 Y0 Z0
N0030 G92 X-10. Y5. Z0
N0040 T1 M6
N0050 G0 G90 X-0.25 Y-0.25 Z0. S1200 M3
N0060 G43 Z.1 H1
N0070 M8
N0080 G1 Z-.25 F6
N0090 Y2.5
N0100 G2 X2.5916 Y4.3143 I2 J0
N0110 G1 X5.2364 Y 3.0875
N0120 G2 X4.5 Y-0.25 I-0.7364 J-1.5875
......
```

(a)

```
line number    : 90
x              : -0.25
y              : 2.5
z              : -0.25
distance       : 0 (absolute)
length units   : 1 (inches)
feed mode      : 0
feed rate      : 6 (ipm)
motion mode    : 10 (linear interpolation)
speed          : 1200
spindle        : 2 (clockwise)
flood          : 1 (on)
tool           : 1 (tool number)
```

(b)

Fig. 3. CNC code example and one of its machine states.

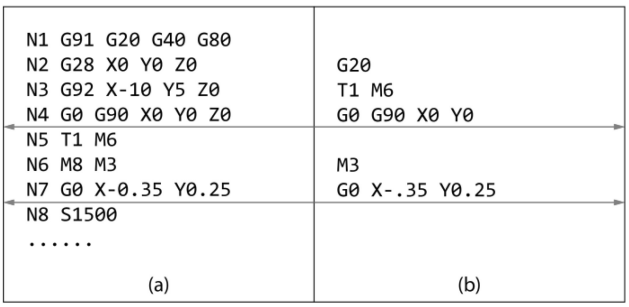

Fig. 4. An example of behavior alignment.

standard. It is designed for a three-axis CNC machine. The axes are X, Y, and Z, which form a right-handed coordinate system of orthogonal linear axes. Algorithm 1 in the appendix describes the overall process of the CNC interpreter. The preprocessing stage removes meaningless characters and formats the uploaded code. The interpreter will then read the code and save the information to a data structure called Block. Block contains all the information needed for the CNC simulator to execute the code. As the interpreter reads the code, it checks the syntax. If it detects a syntax error in a function, it will save the error message and its function name and return the error code. In addition to the NIST RS274NGC, our CNC interpreter can record the machine state as well as the trajectories after the execution of each line of code.

3.3 Hint generation

During the problem-solving process, the students can request hints by clicking the "hint" button. The system will then find the code in the database with the closest match in terms of Behavior & Machine-state distance. If the behavior and machine state of the current code is similar to past code, the next line of the machine state of the past code is used to generate a hint. For each different machine state, a natural language hint is generated. Because CNC programming is highly domain specific, we use a template-based method to generate natural language hints. For example, Fig. 5 shows the template for a deep hole operation (G83).

Algorithm 2 in the Appendix describes the hint generation procedure and Fig. 6 shows an example of how hints are generated. In Fig. 6, (a) and (b) are the current student's code and the historical code, respectively; (c) is the machine state of the last line of the code in (a); (d) is the machine state of the 9th line of the code in (b); and (e) is the generated hints based on the machine state difference between (c) and (d).

3.4 Feedback generation

After the student submits CNC code, the exercise model will call the CNC interpreter to execute the

```
<template>
    Apply deep hole operation at (_x_, _y_) , to
    (_z_), with return distance (_Rn_), and peck (_Qn_).
</template>
```

Fig. 5. Hint generation template for deep hole operation (G83).

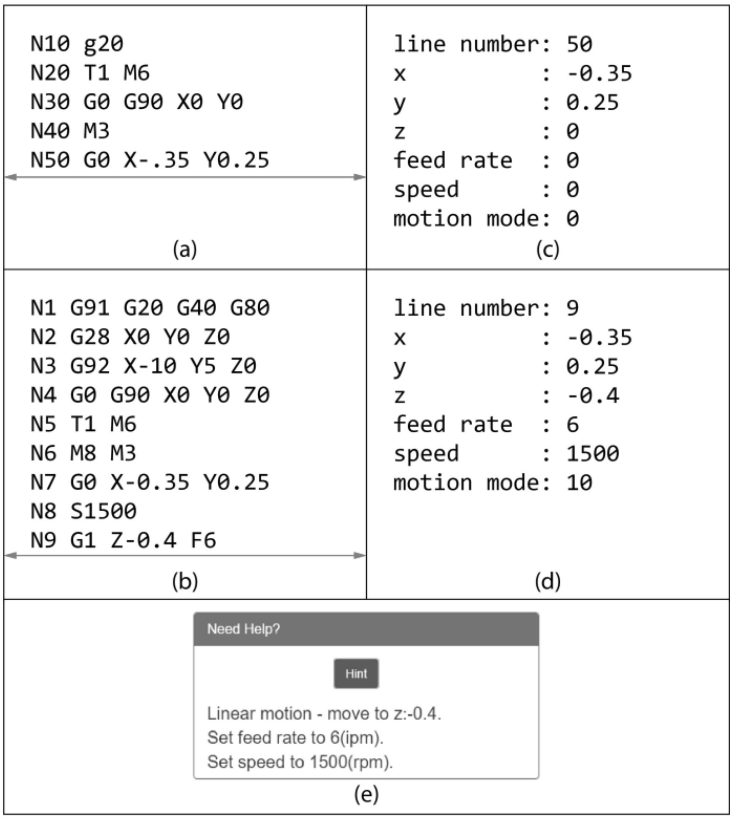

Fig. 6. Hint generation process

Table 2. Instructional effectiveness analysis results

	Test statistic	Critical value	Conclusion
Group 1 (n = 51)	–10.26	2.01	Reject null hypothesis
Group 2 (n = 42)	–6.99	2.02	Reject null hypothesis

code and judge whether the result matches the requirement. An incorrect submission may be caused by syntax error or behavior error. The exercise will generate a candidate list of instructional content, which might contain the concepts or facts that the student missed.

Associated with the label of an incorrect CNC code in the database is a list of instructional content that can help the student to fix the error in the code. Assuming the database contains some labeled data, the exercise model will then search the past labeled data to find the most similar code based on the Behavior & Machine-state distance metric and use the labels to rank the candidate list. The goal is to rank the most related learning content as high as possible, because the students might only click the items near the top. If multiple codes have the same Behavior & Machine-state distance as the current code, the system will vote to rank the candidate list. Then the feedback list is returned to the student as feedback. Algorithm 3 in the Appendix describes the procedure of the feedback generation process.

Our assumption is that the instructional content in the learning module contains all the information the students need to master to be able to solve the exercise problems. The students can click on the feedback items to review the learning content and then come back to solve the problem.

4. Evaluation

The system was evaluated in five ways: pre-test and post-test; accuracy of the generated hints and feedback; learning style evaluation; and the students' opinion survey.

4.1 Pre-test and post-test

There were 93 students who participated in the evaluation of the CNC-Tutor. We designed a pre-test and a post-test that focused on CNC concepts and facts, as well as CNC programming. The students first took the pre-test. After completing the pre-test, they were asked to go through selected topics in the CNC-Tutor, and try to write CNC code to solve an exercise problem. The average time spent with the CNC-Tutor was about one hour. Students were then asked to take the post-test. Based on the students' initial knowledge of CNC, they were divided into two population groups: the first group (Group 1) had 51 students who had limited knowledge of CNC while the second group (Group 2) had 42 students who had some basic knowledge of CNC. An analysis was performed to compare the two groups. The paired T-test results revealed that the null hypothesis was rejected for both groups of students. This suggests that CNC Tutor use resulted in significant improvement in learning.

Table 2 summarizes the test statistics, critical value and conclusions for each test, where the null hypothesis is $\mu_d = 0$, sample sizes were 51 and 42, and the α value is 0.05.

For Group 1, the average score on the pre-test and the post-test were 30.08 and 62.45, respectively. For Group 2, the average score on the pre-test and the post-test were 37.45 and 62.64, respectively. The two groups had similar post-test scores, while Group 2 had higher pre-test score because they already had basic knowledge of CNC programming. The post-test scores for both groups—62.45 (Group 1) and 62.64 (Group 2) were almost identical. This may imply that CNC Tutor is efficient in teaching students the subject matter regardless of prior background. The post-test scores for both groups were about 63 out of 100 possible points. This suggests that the pre-tests and post-tests may have been too difficult and different from the exercises presented in CNC Tutor.

4.2 The accuracy of the generated hints

The hint generation approach was evaluated in two ways. First, we had 42 students use CNC-Tutor to write code for an exercise. Whenever the students requested hints, the system recorded their code and the generated hints. Of the 220 hints recorded, we found 162 hints were meaningful: the accuracy of the hint generation approach was 73.6%. Most of the failed cases were due to the students choosing cutting paths or parameters that the database did not have yet. This can be improved as the database records additional correct solutions.

Because the hint generation approach is data-driven, the more data the database has, the higher accuracy can be achieved. Therefore, the second evaluation approach focuses on the adaptability of the hint generation approach. We divided 38 students into two groups, each with 19 students. The students were asked to write CNC code to cut along a simple path. To have a better code coverage, we generated hints for each line of their code. For the first student group, the database contained only five

correct sample codes, which were used to generate the hints. For the second group, the database contained the five sample codes plus the codes from the first group. It is noteworthy that the code in the second group contained both correct and incorrect codes. Our approach generated 50 meaningful hints out of 79 (65.8% accuracy) for the first group, and 66 meaningful hints out of 74 (89.2% accuracy) for the second.

The feedback generation approach requires labeled data in the database. We took 60 samples of the students' code to evaluate the feedback generation approach. We labeled 30 of the codes and used the other 30 for evaluation. We treated feedback generation as a ranking problem, because our goal is to help the students find out missing facts or misconceptions by ranking the most helpful learning content as high as possible. Therefore, we used normalized discounted cumulative gain (NDCG) to evaluate the performance of our feedback generation approach. The calculation of NDCG is shown below, where p means the number of items; rel_i means the importance of item i; and IDCG means ideal DCG value.

$$NDCG = \frac{DCG}{IDCG}$$

$$DCG = \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}$$

We calculated NDCG@1 and NDCG@3 (NDCG@n means the NDCG value for the first n items), because during the evaluation by the students, we observed that most of them only clicked on the first three items in the feedback list. To calculate the NDCG score, we set the importance of the first three relevant items as 3, 2, and 1. The value of NDCG@1 and NDCG@3 were 0.6 and 0.662, respectively.

4.3 Learning style analysis

Among the 93 students who participated in the pre-test and post-test evaluation, 40 also participated in a survey of learning styles [19]. The learning styles have four dimensions: (1) active (ACT) or reflective (REF), (2) sensing (SEN) or intuitive (INT), (3) visual (VIS) or verbal (VRB), and (4) sequential (SEQ) or global (GLO). Active learners prefer to get involved in experiments, while reflective learners prefer to think about the information first. Sensing learners like using well-established methods to solve a problem and are patient in learning new facts, while intuitive learners usually are innovative in the problem solving process, and dislike repetitive work. Visual learners prefer to learn by viewing diagrams, demonstrations, and so on, while verbal learners prefer to learn through text and dialogue. Sequential learners tend to absorb new information sequentially, while global learners tend to choose instructional content in the order they like.

We analyzed the correlation coefficient between learning style and learning gain in the pre-test and post-test. For each dimension of the learning style, we set a value from –11 to 11. For the four learning styles, the correlation coefficients were –0.21, –0.13, –0.23, and 0.09, respectively. The result shows that active learners and visual learners tend to benefit more from CNC-Tutor, compared with reflective learners and verbal learners.

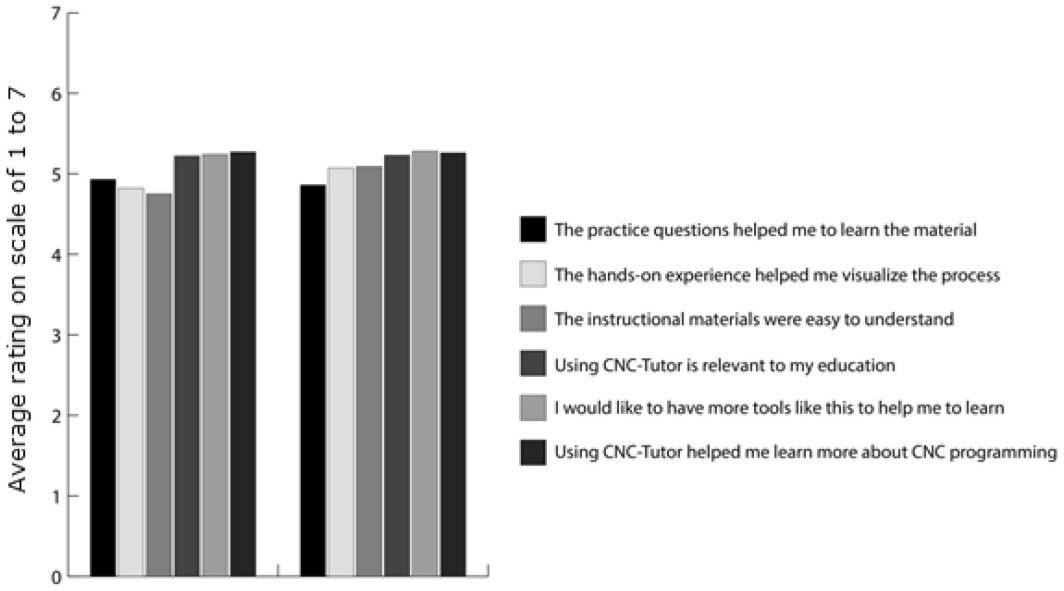

Fig. 7. Average ratings on student opinion survey; group 1 is on the left (limited CNC knowledge), and group 2 is on the right (basic CNC knowledge).

4.4 Opinion survey

There were 93 students who completed an opinion survey about the system, illustrated in Fig. 7. As described above, students were in two groups based on prior CNC knowledge. The students in the first group had limited knowledge of CNC while the students in the second group possessed basic knowledge of CNC. The major difference of the ratings was on the third question: The instructional materials were easy to understand, for which the first group gave only 4.75 points, while the second group gave 5.09. Overall, the survey of the CNC-Tutor shows a positive impact of CNC-Tutor on the students' learning experience.

4.5 Discussion and lessons learned

These results suggest that the CNC Tutor design is instructionally effective, and that students' subjective impressions of the system are positive; however the average post-test score was around 62 out of 100. Perhaps the test questions need to be modified to better align with CNC Tutor's test bank or a greater variety of practice questions is needed in the tutor test bank. It appears that we may safely continue to develop similar types of Intelligent Tutoring Systems for other engineering subjects. It also appears that CNC Tutor's explanations and feedback are a good fit for active, visual learners. Possible enhancements include the addition of more video and/or simulation tools to help learners visualize abstract concepts.

5. Conclusions

We built an Intelligent Tutoring System for the domain of Computer Numerical Control, especially CNC programming. Our main observation is that students fail to solve CNC programming problems mainly due to two reasons: lack of problem solving skills and misconceptions. In CNC-Tutor, the learning module and the quiz module focused on teaching facts and concepts, while the exercise module focused on improving the student's problem solving skills. We proposed algorithms to interpret user's CNC code, and then provide hints and generate feedback based on past performance data.

Experiments showed that the developed approach can generate meaningful hints over 85% of the time, and the feedback can help the students find out missing facts and misconceptions effectively. The pre-test and post-test results and opinion survey show a positive impact on students' learning gains and experience in learning CNC programming. A correlation coefficient suggests that CNC-Tutor had a stronger influence on learning outcome for active and visual learners.

Currently, CNC-Tutor contains only basic facts, concepts and CNC programming knowledge. More in-depth lectures and exercises can be added to the CNC-Tutor. Because analyzing and labeling CNC code is labor-intensive, we may consider using a crowdsourcing approach that allows students themselves to provide help messages. Future plans also include continuing to develop intelligent tutoring systems for engineering applications and embedding animations, simulations, and/or videos into future system designs to make abstract concepts easier to grasp.

Acknowledgment—This material was supported by a National Science Foundation Advanced Technology Education Program grant (No. 1304843) and a gift from Rockwell Automation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation or Rockwell Automation.

References

1. V. Petrushin, Intelligent tutoring systems: architecture and methods of implementation. A survey, *Journal of Computer & System Sciences International*, **33**, 1995, pp. 117–139.
2. V. Tsiriga and M. Virvou, A Framework for the Initialization of Student Models in Web-based Intelligent Tutoring Systems, *User Modeling and User-Adapted Interaction*, **14**, 2004, pp. 289–316.
3. A. C. Graesser, M. W. Conley and A. Olney, *Intelligent tutoring systems*, APA Educational Psychology Handbook, 3, 2012.
4. S. J. Hsieh and Y.-T. Cheng, Algorithm and intelligent tutoring system design for programmable controller programming, *The International Journal of Advanced Manufacturing Technology*, **71**, 2014, pp. 1099–1115.
5. C. J. Butz, S. Hua and R. B. Maguire, A web-based intelligent tutoring system for computer programming, in Web Intelligence, 2004, WI 2004, *Proceedings, IEEE/WIC/ACM International Conference on*, ed, 2004, pp. 159–165.
6. I.-H. Hsiao, P. Brusilovsky and S. Sosnovsky, Web-based parameterized questions for object-oriented programming, in *Proceedings of World Conference on E-Learning, E-Learn*, ed, 2008, pp. 17–21.
7. P. Brusilovsky and S. Sosnovsky, Individualized exercises for self-assessment of programming knowledge: An evaluation of QuizPACK, *Journal on Educational Resources in Computing (JERIC)*, **5**, 2005, p. 6.
8. S. Abu-Naser, A. Ahmed, N. Al-Masri, A. Deeb, E. Moshtaha and M. AbuLamdy, An intelligent tutoring system for learning java objects, *International Journal of Artificial Intelligence and Applications (IJAIA)*, **2**, 2011.
9. W. Jin, T. Barnes, J. Stamper, M. J. Eagle, M. W. Johnson and L. Lehmann, Program representation for automatic hint generation for a data-driven novice programming tutor, in *International Conference on Intelligent Tutoring Systems*, ed, 2012, pp. 304–309.
10. K. Rivers and K. R. Koedinger, A Canonicalizing Model for Building Programming Tutors, in *Intelligent Tutoring Systems: 11th International Conference, ITS 2012*, Chania, Crete, Greece, June 14–18, 2012. Proceedings, Berlin, Heidelberg, 2012, pp. 591–593.
11. T. Lazar and I. Bratko, Data-Driven Program Synthesis for Hint Generation in Programming Tutors, in *Intelligent Tutoring Systems: 12th International Conference, ITS 2014*, Honolulu, HI, USA, June 5–9, 2014. Proceedings, Cham, 2014, pp. 306–311.
12. N.-T. Le and W. Menzel, Using Weighted Constraints to Diagnose Errors in Logic Programming-The Case of an Ill-

defined Domain, *International Journal of Artificial Intelligence in Education*, **19**, 2009, pp. 381–400.
13. K. Rivers and K. R. Koedinger, Automatic generation of programming feedback: A data-driven approach, in *The First Workshop on AI-supported Education for Computer Science (AIEDCS 2013) 50, ed, 2013.*
14. A. Overby A, CNC machining handbook: building, programming, and implementation: McGraw-Hill, Inc., 2010.
15. J. Valentino and J. Goldenberg, Introduction to computer numerical control: Pearson, 2012.
16. A. Mitrovic, B. Martin and M. Mayo, Using Evaluation to Shape ITS Design: Results and Experiences with SQL-Tutor, *User Modeling and User-Adapted Interaction*, **12**, 2002, pp. 243–279.
17. S. Gross, B. Mokbel, B. Hammer and N. Pinkwart, Feedback provision strategies in intelligent tutoring systems based on clustered solution spaces, *DeLFI 2012: Die 10. e-Learning (S. 27–38)*
18. T. R. Kramer, F. M. Proctor and E. Messina, The NIST RS274NGC interpreter-version 3, 2000.
19. R. M. Felder, L. K. Silverman and others, Learning and teaching styles in engineering education, *Engineering Education*, **78**, 1988, pp. 674–681.
20. V. Aleven, B. M. McLaren, J. Sewall and K. R. Koedinger, A new paradigm for intelligent tutoring systems: example-tracing tutors, *International Journal of Artificial Intelligence in Education*, **19**(2), 2009, pp. 105–154.
21. S. J. Hsieh and P. Y. Hsieh, Animations and intelligent tutoring systems for programmable logic controller education, *International Journal of Engineering Education*, **19**(2), 2003, pp. 282–296.
22. R. R Choudhury, H. Yin and A. Fox, Scale-driven automatic hint generation for coding style, *13th International Conference on Intelligent Tutoring Systems (ITS 2016)*, 9684, 2016.
23. T. W. Price, R. Zhi and T. Barnes, Hint generation under uncertainty: the effect of hint quality on help-seeking behavior, *18th International Conference on Artificial Intelligence in Education*, 2017, pp. 311–322.
24. 24. A. Ogan, E. Yarzebinski, P. Fernández and I. Casas, Cognitive tutor use in Chile: understanding classroom and lab culture, *17th International Conference on Artificial Intelligence in Education*, 2015, pp. 318–327.
25. H. A. A. Hasanein and S. S. A. Naser, An intelligent tutoring system for cloud computing, *International Journal of Academic Research and Development*, **2**(1), 2017, pp. 76–80.
26. D. Hooshyar, R. B. Ahmad, M. Wang, M. Yousefi, M. Fathi and H. Lim, Development and evaluation of a game-based Bayesian intelligent tutoring system for teaching programming, *Journal of Educational Computing Research*, **56**(6), 2018, pp. 775–801.
27. C. S. González, P. Toledo and V. Muñoz, Enhancing the engagement of intelligent tutorial systems through personalization of gamification, *International Journal of Engineering Education*, **32**(1), 2016, pp. 532–541.

Appendix

Algorithm 1
CNC_interpreter_simulator(code)
  Pre–process (code)
  for each line of code
    Read the code and save the information into Block
    if there exists syntax errors
      return the error code
    end
    Call CNC simulator to execute
    return the trajectories and the machine states
  end

Algorithm 2
Hint_generation(code A)
  CNC_interpreter_simulator(code A)
  for each current code in the database
    Calculate the distance with A
  end
  Find the code (B) with the smallest distance with A
  Calculate the machine state difference between A and B
  Generate natural language hints using the templates

Algorithm 3
Feedbackgeneration(code A)
  result, machine state = CNC_interpreter_simulator(code A)
  if result is incorrect
    foreach historical code B in database
      distance = calculate_distance(A, B)
    end
    similar code = K_Nearest_Neighbors(distance)
    feedback list = get the feedback label from the similar code
    return the feedback list
  End

Qinbo Li was a Master of Science student in the Department of Computer Science and Engineering at Texas A&M University at the time of this work. He is now a PhD student at Texas A&M University, advised by Dr. Yoonsuck Choe. His research interests are Artificial Intelligence, Machine Learning, and Computer Graphics.

Sheng-Jen ("Tony") Hsieh, PhD is a Professor in the College of Engineering at Texas A&M University. He holds a joint appointment with the Department of Engineering Technology and the Department of Mechanical Engineering. His research interests include engineering education, cognitive task analysis, automation, robotics and control, intelligent manufacturing system design, and micro/nano manufacturing. He is also the Director of the Rockwell Automation laboratory at Texas A&M University, a state-of-the-art facility for education and research in the areas of automation, control, and automated system integration.