Online Tutoring to Support Programming Exercises

Bryn Jeffries
Grok Learning
Sydney, New South Wales, Australia
bryn@groklearning.com

Marion Zalk
The University of Melbourne
School of Computing and Information Systems
Melbourne, Victoria, Australia
mzalk@unimelb.edu.au

Timothy Baldwin
The University of Melbourne
School of Computing and Information Systems
Melbourne, Victoria, Australia
tbaldwin@unimelb.edu.au

Ben Taylor*
Grok Learning
Sydney, New South Wales, Australia
ben@groklearning.com

ABSTRACT

Programming exercises lend themselves to learning environments that can provide automated testing and formative feedback to allow students to learn independently and at their own pace. Even so, there are still situations where there is no substitute for interaction with a human expert to provide guidance and encouragement to students. We describe how online tutoring is used within two very different contexts on the same platform. While both contexts use a specific learning environment, many of the experiences encountered in providing online tutoring support provide important lessons for educators that generalise to using any comparable environment.

CCS CONCEPTS
• Social and professional topics → CS1; Student assessment; K-12 education.

KEYWORDS
computer science education, programming assignments, online education, automated feedback, tutoring

ACM Reference Format:

1 INTRODUCTION

Like all learning activities, a major component of learning to program is through practice by the learner. Since many programming activities can be designed to have testable outcomes, many learning environments have been designed to automate the task of assessing student submissions [6, 18]. The analysable structure of code and how it runs has additionally afforded the development of visualisations, next step hints, and recommendations [17]. Such systems provide a means to reduce the burden on teaching staff, increase autonomy of students in progressing through exercises, and teach online at scale. Even so, it is our firm conviction that students still benefit from having a means to communicate with teaching staff while working on programming exercises.

1.1 Classroom Tutoring

In most university and school teaching, students exclusively communicate with teaching staff in a classroom. Although the classroom environment is popular, it does have its problems. In the authors’ experience, when students are engaged in programming exercises they often need individualised help, encounter different problems, and move at very different paces. Since classroom time is limited, students are typically expected to continue their work outside classroom time. It is often during this unsupervised time that a student engages with a problem, and is thus more likely to come across issues. The risk is that these issues remain unresolved until the student’s next scheduled class, at which point it is too late for the student to obtain and make use of feedback. In essence, students don’t always encounter problems at a convenient time for the classroom schedule.

1.2 Individual Tutoring

Individual tutoring can help with some of these problems. Individualised help and focused time can help push students towards points of struggle more quickly. In comparison to classroom tutoring, individual tutoring has been shown to substantially improve learner understanding and attitudes [4]. Students involved in individual tutoring ask more questions (about 240 times more) than in a classroom environment [8] and in primary schools it has been shown that even volunteer tutors can improve learning [15]. While these are useful findings, it is difficult to include substantial amounts of individual tutoring at the scale of a university, or a school environment.
1.3 Online Tutoring

Online tutoring, delivered through an online learning environment, presents another option for communication with students. The majority of the literature on online tutoring research appears to focus on the tutor as a group facilitator, or coach, similar to the way they act as a classroom tutor [2, 5, 7, 10, 13], which is subject to some of the problems of traditional classroom teaching. While massive open online courses (MOOCs) appear to be an interesting space to explore online tutoring, these environments tend to favour a scalable discussion forum format [3, 12, 14], where communication is driven by students and tutors are not essential. A counter to this theme is Codeopticon [9], a prototype interface built for scalable individual online tutoring. This sort of online approach affords some interesting differences to a classroom: tutors are able to help in their own time and they are able to tutor multiple learners simultaneously. This allows tutors to be available when students are struggling, while also being more efficient with their own time, with the possible downside of lacking the ability to interactively sketch out a schematic solution, or demonstrate a key programming concept.

1.4 Individual Online Tutoring

Individual online tutoring can offer similar benefits to individual in-person tutoring, but with novel affordances provided by the online tutoring environment. It has the promise of achieving educational benefits similar to individual tutoring [4] along with the scalability of online tutoring [2, 7, 9, 10]. A potential deficiency of individual online tutoring is that, much like individual tutoring, it is unable to scale when a tutor is required to communicate with every member of a cohort. This limitation can be alleviated if a conversation between a student and tutor is not required to be completed in one continuous session. Without this constraint, tutoring can be asynchronous, in that the conversation happens over a longer period, with gaps of minutes, hours or even days between questions and answers. This time can allow for reflection, but also for batched approaches to question answering. Tutoring can also be multiplexed, in that the tutor is able to maintain conversations with multiple individual students at the same time, exploiting gaps of minutes in conversation that allow for an improvement in efficiency. Together these approaches allow students to get started on an activity at a time that suits them, ask questions when it suits them, and still receive an response earlier than a traditional classroom (where the delay could be a week). For tutors the potential benefit is that they are able to work flexibly in focused batches, delivering assistance to the students who need it most.

This paper presents the authors’ experiences of using online tutoring within a common programming learning environment. The relevant features of the learning environment are described in Section 2. Section 3 then explains how online tutoring is used in two very different contexts: a large-scale programming course for primary and secondary school students, and an introductory programming subject for first-year students at a university. Analysis of the data collected from these two contexts is presented in Section 4.

2 Programming Learning Environment

The case studies described in Section 3 use the same learning environment, Grok Learning (https://groklearning.com/), for their programming classes. The learning environment provides content as a sequence of slides, grouped into modules on distinct topics. The slides in a module can contain content describing the topic, or programming exercises where a student can apply their understanding of the earlier content slides. Ideally the modules are sufficiently self contained that a student has all the information to hand to be able to answer the problem.

Each problem requires the student to write or modify code, which is then submitted for marking. As with many other autograding systems, this marking is principally performed through automated comparisons of output against expected results for various input conditions. Students can be immediately informed of the results of tests, such as a description of the difference to expected output, allowing them to refine their code and resubmit. Students can also run their code within the learning environment unlimited times without penalty, prior to submitting for marking. Marks are available for each problem, and various scoring rules can be used, such as reducing the maximum available points based on the number of submissions.

The learning environment provides a text editor with syntax highlighting, and problems can incorporate an arbitrary number of source and data files. An example of a problem displayed in the learning environment is shown in Figure 1, with key elements of the interface annotated.

2.1 Online Tutoring Facility

The learning environment provides an online tutoring facility to support programming activities, and this facility has evolved to meet the needs that have arisen through using it in a variety of courses. The user interface for attempting problems features a button that a student can click to open a chat window to ask questions,
as shown in Figure 1. Tutors can monitor for new messages via a triage page, and volunteer to respond to the student. Additionally, students deemed to be struggling (by having made 5 unsuccessful submissions and not yet requesting assistance) are flagged in the triage page, and tutors can choose to initiate a chat with the student.

All messages posted by the student or tutors belong to a thread associated with the student and the problem from which the chat window was opened. Tutors involved in a discussion with a student are able to see the context of the problem, including the problem description, the student’s code history, as well as teacher notes or example solutions that can provide useful prompts to give to the student. An example of the tutor’s view is shown in Figure 2. As well as posting messages to the student, tutors can leave messages visible only to other tutors, allowing suggestions or important details to be preserved, such as when a tutor hands over to another tutor at the end of a shift. The tutor can also modify and test a sandboxed version of the student’s code without affecting the student’s own submissions, to allow them to diagnose problems and experiment with suggestions.

Since students often ask questions about common issues, responses can be saved and re-used, with template parameters to personalise the response for the student. These saved responses can be generic for use in other problems (such as guidance on debugging a function) or associated with a specific problem. Senior tutors also have the power to lock a student’s account to prevent them from using chat in cases where the student is abusive to the tutor.

3 CASE STUDIES
Here we present the use of online tutoring in two different contexts: a large online course for primary and secondary school children, supported by volunteer tutors; and a large introductory programming subject at a university. Both of these activities are relatively mature, having used the same platform and provided online tutoring support over several years.

3.1 School Programming Course
The National Computer Science School (NCSS) is an educational outreach program [16] that offers an annual programming course, the NCSS Challenge, using the Grok Learning learning environment. The course targets Australian school students in years 5–10. Students could register directly or through their school, and some teachers required their students to work on the problems as part of their classroom or homework activities. The creation of the course’s content has been guided by the principle that it should be open to novice programmers, with no assumed knowledge.

Several streams were provided in the 2018 offering, allowing students (or their teachers) to choose the level most appropriate to their abilities. More introductory streams used Blockly as the programming language, while more advanced streams used Python. These streams were ‘Newbies’ (Blockly, years 5–6), ‘Beginners Blockly’ (Blockly, years 7–8), ‘Beginners Python’ (Python, years 7–8), ‘Intermediate’ (Python, years 9–10), and ‘Advanced’ (Python, years 10+). Each stream’s material was aligned to the corresponding years’ Australian Digital Curriculum [1] and was designed to gradually acquaint the student with programming concepts, with working examples, and incremental problems with low barriers to attempting them.

In the 2018 offering, new content was released each week, for five weeks, consisting of explanatory slides on that week’s topics, along with several programming problems relating to those topics. Since the course is primarily designed as a learning task rather than a formal assessment, each problem provided scaffolding to help students reach a solution, including notes containing all necessary information, working examples, hints and submission feedback. The number of problems to complete each week varied by stream, with eight for Newbies and both Beginners, and five for Intermediate. The Advanced stream had more complex problems, with five smaller problems in the first week tapering to two larger problems in each of the final two weeks. Each week’s material had a two week window to complete before submissions were marked late (and received no score), except for the final week’s module which was available for just one week.

To support the students, 170 volunteer tutors were recruited to respond to help requests via the live tutor interface. All tutors were selected after a formal review process, and came from a variety of backgrounds including tertiary students, industry professionals, and high-school computing teachers. Usually two or three tutors were rostered to be available 8 am–9 pm each day, including weekends. All tutors were subscribed to a messenger service (Slack: https://slack.com) to communicate with each other, allowing additional tutors to be alerted when help was required to deal with large numbers of student help requests. Tutors also attended a training session, prior to the start of the course, on how to use the live tutor interface and give appropriate advice to students.

Because most students taking part in the course were children, a number of safety measures were put in place. All tutors were required to complete a Working with Children police check as part of the application process. A paid adult member of staff was on duty during all rostered hours to handle escalation of any serious issues, such as use of abusive language by the child or comments...
that indicated that a child’s safety may be at risk, in which case the student’s class teacher would be informed.

While the online tutoring facility has been available in previous years’ offerings, usage required the students to notice the tutoring icon within the learning interface and try clicking it. From earlier years’ survey responses there was anecdotal evidence that school students often aren’t aware that the facility was available. To improve awareness in the 2018 course, students and their teachers were contacted by email to inform them of the service prior to the start.

3.2 University Subject

The introductory programming university subject (COMP10001: Foundations of Computing, the introductory computing subject for first-year undergraduates at The University of Melbourne) consists of 3 hours of lectures and one 2-hour workshop each week over 12 weeks, and is based primarily on Python. The first hour of the workshop takes the form of a tutorial, with written materials hosted outside the programming learning environment which are to be completed without the use of a computer. During the second hour of the workshop, students are based in a computer lab and work individually on worksheets hosted within the learning environment, with roaming tutors providing in-person assistance. There are weekly due dates for the mark calculation of worksheets, spaced through the semester, but students can complete the worksheets at any time prior to the due date, with no limit on the allowable number of attempts (and no impact on the mark based on the number of attempts). The number of worksheets and questions within a worksheet varies from week to week. They are designed to align with the content of lectures and tutorials, by providing practical problems of increasing complexity for students to learn both basic programming constructs and the specifics of Python, as well as progressively develop programmatic problem-solving skills. In addition to the worksheets, the subject also has three programming projects, all of which are also hosted in the programming learning environment.

As the subject is introductory, students are expected to have little or no programming experience at the start of the subject, and this is reflected in the content. Most students who enrol in the subject are in their first semester of an undergraduate degree (primarily in the Bachelor of Science), but some students take the subject later in their undergraduate degree, generally as a “breadth” subject. The time commitment for students to complete the worksheets and projects differs, dependent on individual student ability. While some students complete all worksheets and projects in labs, the vast majority spend significant time working on them outside the labs. Sample solutions for all worksheets and projects are made available after the deadline has passed. Together, all the weekly worksheets constitute 10% of the final mark for the subject, and the projects contribute a combined total of 30%.

In addition to face-to-face assistance in workshops, there is a weekly roster for tutors to monitor the subject forums and also individual help requests via the tutor messaging interface. Students are instructed to use the discussion forums for any questions that do not involve code that is part of the subject assessment, and to use tutor messaging for questions that relate to code for assessed work. The number of tutors is directly proportional to the number of students enrolled in the semester, so the number of hours where help requests are actively monitored is dependant on the size of the student cohort. In semester 1, 2018 tutor messaging was monitored for >10 hours per day, and in semester 2, 2018, it was monitored for >5 hours per day, in each case spread across 8 am–12 am each day.

In addition to periodic (asynchronous) monitoring of help requests, particular time slots are nominated for “live” (i.e., semi-synchronous) help via the tutor messaging interface each week day. These times are advertised to students, and they are encouraged to use this as a means to access additional help, especially in situations where more interactive help is required. This occurs in teaching weeks at lunch-time and in the early evening. Here, tutors prioritise student questions submitted within the time slot, over questions submitted previous to it.

The tutor cohort is quite diverse, in that tutors can be undergraduates, postgraduates, or professionals who have completed their studies. All tutors have completed at least a second programming subject, with a high mark. A lecturer monitors tutor messaging, and more experienced tutors also informally monitor tutor responses to ensure that there is consistency and professionalism in the support provided. Tutors are also coached on the fine line between bumping students in the right direction and providing them with the answer verbatim, as it can be quite challenging for some tutors to learn what is appropriate in terms of both face-to-face and online help. Experienced tutors have created a set of “canned” responses for recurring question types, which are available for use by all tutors. The most popular question is students wanting to know “hidden” test cases that are used as part of the automatic assessment of each problem, to test the generalisability of the submitted code. Tutors and the lecturers use a private Slack instance to ensure consistency/correctness in answers provided to students, alert each other of issues with/fixes to specific problems, discuss possible additions to the pool of “canned” responses, or alert the lecturing staff to concerns over specific students (e.g., issues of well-being).

Use of the tutor messaging interface by students varies tremendously. Most students make little use of it, but there is generally a small proportion of very heavy users, especially by students who feel uncomfortable asking face-to-face questions in the lab, or who are ritually attempting to complete the worksheets right on the due date and seeking last-minute assistance to complete problems. The service tends to be used most heavily for projects rather than worksheets, primarily for the following purposes: clarifying specifications or confirming the validity of specific visible test cases (as a reflection of the fact that the projects are generally considerably harder than worksheet problems); seeking general direction with project questions; and requesting assistance in debugging code, which is inevitably more complex in projects than in the worksheets.

1There is also a small amount of HTML covered in the subject, but there are no materials for this in the programming learning environment.

2As distinct from “visible” test cases which are provided as part of the problem specification, where the student is provided with both the input and the correct output to exemplify the problem. For the hidden test cases, the only feedback students get is that their code has failed, not what specific input it has failed on; hence the inevitable requests for hidden inputs, and need to provide responses outlining the pedagogical intent behind them, and general guidelines on how to test/debug code.
Table 1: Student and tutor participation numbers in case studies

<table>
<thead>
<tr>
<th>Stream</th>
<th>Members</th>
<th>Chat users</th>
<th>Tutors</th>
</tr>
</thead>
<tbody>
<tr>
<td>University Subject</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Semester 1</td>
<td>1151</td>
<td>529 (46.0%)</td>
<td>23</td>
</tr>
<tr>
<td>Semester 2</td>
<td>641</td>
<td>270 (42.1%)</td>
<td>19</td>
</tr>
<tr>
<td>School Course</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>newbies</td>
<td>3953</td>
<td>349 (8.8%)</td>
<td>77</td>
</tr>
<tr>
<td>beginners-blockly</td>
<td>2414</td>
<td>339 (14.0%)</td>
<td>74</td>
</tr>
<tr>
<td>beginners</td>
<td>7970</td>
<td>1220 (15.3%)</td>
<td>119</td>
</tr>
<tr>
<td>intermediate</td>
<td>4753</td>
<td>617 (13.0%)</td>
<td>109</td>
</tr>
<tr>
<td>advanced</td>
<td>721</td>
<td>113 (15.7%)</td>
<td>33</td>
</tr>
</tbody>
</table>

*a*Includes all tutors who participated at least once in tutor interface. Tutors may have been active in multiple streams, so numbers are not disjoint.

4 ANALYSIS

The programming learning environment used for the case studies described in Section 3 retains information on the submissions made by students, including the submission time and test outcomes. It also records all online tutor message activity, including the time and author of each post.

This section presents analysis of the data collected for the two case studies, motivated by two main research questions. Firstly, the various ways in which students make use of the tutoring facility are explored in Section 4.1. Section 4.2 extends this analysis by examining the timing of problem attempts and requests for tutor help. Finally, the impact of online tutoring tutoring is explored in Section 4.3 by examining the final outcomes of problem attempts, in relation to online tutor interactions.

4.1 Chat usage

The numbers of students taking part in the university subject and school course are shown in Table 1. In the university subject, despite the differences in size of the cohorts in the two semesters, usage of the live chat was consistent at around 45%. In the school course, chat usage was between 13–16% in all streams except “Newbies”, which was substantially lower at 9%. This lower usage may reflect the younger audience finding it harder to use the feature because of the literacy demands of articulating their problem in words and the challenge of typing.

Except where stated otherwise in the analysis that follows, results are aggregated across all cohorts (i.e., across both semesters for the university subject, all streams of the school course).

4.1.1 Thread count. The level of usage can be further refined by counting the number of threads each user created (each thread corresponding to a different problem attempted by the student). About 45% of students used the live tutor interface at least once in the university subject, across both semesters. Figure 3 shows that, of these, most students used live chat once or twice, but some (about 4% of all students) used it for more than 10 problems. For the school course, of those students who used live chat, the number of threads per student was lower than for the university subject, and this was consistent across all 5 streams despite the differences in overall usage shown in Table 1.

4.1.2 Chat thread length. Chat threads between a student and tutors can range from very short clarifications to extended dialogues involving multiple tutors over several days. The simple analysis of threads presented here quantifies thread length by counting the number of individual exchanges between a student and any tutor, by aggregating all consecutive posts from the student as a single run. Multiple consecutive posts from any tutors and also treated as a single run. In the university subject, the majority of problems are resolved with one or two exchanges between the student and tutor, as can be seen in Figure 4. Very few threads involved more than 10 exchanges. Many of the single-run threads are due to students solving the problem for themselves while awaiting a response from the tutor, and in such circumstances the student may have still benefited from the live tutor system through rubber-duck debugging — whereby articulating a problem to a third party often reveals a solution without requiring a response [11].

In the school course the more junior streams had a significant proportion of threads that received no initial reply, predominantly because they were initiated by the tutor to students identified as struggling, and the student did not respond. All streams included
cases of long-running threads, with the intermediate and advanced streams having the highest proportions of threads of more than ten exchanges. From inspection of several cases, this appears to be due to students working in a stream beyond their capabilities, and becoming dependent upon the tutor to work through a problem. This effect was exacerbated by the higher availability and lower response times of the tutors compared to the university subject.

4.1.3 Thread length vs. thread count. Figure 5 compares the count to the median length of all threads for each student. It can be seen that students who used live chat for many problems tended to have quite short conversations with the tutor, while students who had very long conversations had fewer of them. The former group could be thought of as efficient chat users, though reliant on frequent help, while the latter group seem to require a lot of help on a small number of questions.

4.2 Activity times

For each problem attempted by a student, times are recorded for all code submitted for automatic marking, and for any chat posts between the tutor and student associated with the problem. To properly examine the influence of the chat facility on the way in which students attempt questions would require a controlled experiment. Even so, it is interesting to look at the behaviour of students who explicitly requested help, and when this help was requested.

4.2.1 First attempt times. Figure 6 shows the distribution of first attempt submissions according to time of the week, with separate distributions shown according to whether the student also requested help for that problem. The university subject weekly deadline of end of Monday can be seen to result in a peak in first question attempts on the final day, but there is also a steady number of first attempts throughout the rest of the week. The large peak just prior to the deadline among students who don’t request help suggests these students realise they have left their attempt too late to expect a response from tutors. Promoting earlier starts on questions may also encourage students who would benefit from help to seek it.

In the school course it can be seen from Figure 6b that most first attempts are towards the start of the week, with few started over the weekend, which would be in keeping with the student attempting these problems in the classroom. Of those attempts for which help was requested, the surge prior to the Sunday deadline suggests that students working on questions at home are an important group of users of the live tutor support.

4.2.2 Help request times. Figure 7 shows the times of day for the initial requests for help. In the university subject, despite students being informed of the rostered live tutor hours of 1–2 pm and 7–8 pm, it can be seen in Figure 7a that students submitted questions throughout the day, particularly in the afternoon and evening. By
contrast, students in the school course made full use of the rostered window of 8 am–9 pm each day, as shown in Figure 7b. During the week, activity was greatest during school classroom hours and lower in the evening, while in the weekend this activity profile was reversed.

4.2.3 Response times. In the university subject, although tutors often voluntarily responded to questions in their free time, the majority of cases were handled during the rostered live tutor periods. Since students often posted outside of these periods, there was often a long delay in the tutors’ responses, as can be seen from Figure 8a. The bulge between 12 and 18 hours delay can thus be explained by students posting in the evening and receiving a response the following day. By contrast, the higher availability of tutors in the school course yielded much shorter first response times, as shown in Figure 8b.

4.3 Problem outcomes

Figure 9 shows the numbers of students attempting each problem, divided into whether the attempt resulted in success or failure, and whether the student’s attempt involved interaction via online tutoring. For the university subject the data is limited to worksheet problems, and the problems are grouped by worksheet in Figure 9a. For the school course all problems are shown in Figure 9b, grouped by week. In both cases it can be seen that the vast majority of problems attempted result in a successful outcome (even if many submissions may have been required to get to that outcome). By far

Figure 7: Histogram of time of day of initial requests for help by students. Values are daily average across week days and weekend days.

Figure 8: Histogram of response times, calculated as time from first post from the student until the first response from a tutor. Note the different time-scales for each case.

Figure 9: Problem attempt outcomes. Each bar represents a single problem.
Table 2: Outcomes for problems, grouped by level of online tutor interaction

<table>
<thead>
<tr>
<th>Outcome</th>
<th>University subject</th>
<th>School course</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Solved</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>no tutor interaction</td>
<td>129,574 (95.8%)</td>
<td>344,669 (96.2%)</td>
</tr>
<tr>
<td>after response^a</td>
<td>1,911 (1.41%)</td>
<td>4,116 (1.15%)</td>
</tr>
<tr>
<td>before response^b</td>
<td>751 (0.56%)</td>
<td>699 (0.19%)</td>
</tr>
<tr>
<td>after prompt^c</td>
<td>92 (0.07%)</td>
<td>516 (0.14%)</td>
</tr>
<tr>
<td><strong>Failed</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>no tutor interaction</td>
<td>2,701 (2.00%)</td>
<td>7,828 (2.18%)</td>
</tr>
<tr>
<td>after response^a</td>
<td>216 (0.16%)</td>
<td>350 (0.10%)</td>
</tr>
<tr>
<td>without response^d</td>
<td>12 (0.01%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>after prompt^c</td>
<td>18 (0.01%)</td>
<td>205 (0.06%)</td>
</tr>
</tbody>
</table>

^a Student initiated chat thread, and outcome was after a tutor had provided a responding post
^b Student initiated chat thread but reached a solution before tutor responded
^c Outcome reached after tutor provided unsolicited help
^d Student initiated thread but did not receive a responding post from tutor

the main impediment to students’ final scores was that problems were not attempted.

For the university subject, the first week shows a decline in attempts in later problems that may be attributed to students being less committed to completing work in the first week of their studies, perhaps because they have not decided on their choice of enrolments. The final two worksheets were optional, covering more advanced topics, and the final worksheet in particular can be seen to have only been attempted by a small number of very committed students. It can be seen that fewer students attempted later questions in a worksheet, and the number of students starting worksheets dropped each week. The more challenging questions toward the end of each week’s material can be seen to have higher numbers of instances of success with help via live tutoring, as well as failures with no tutor interaction.

For the school course, with the exception of the advanced stream, the numbers of instances of cases other than success without tutor help are too small to be seen. All streams have a high degree of attrition in number of students attempting later problems, both within each week’s problems and between weeks.

Table 2 shows the different outcomes aggregated over all attempted problems, according to the success or failure of the students’ final submission and the related interactions via online tutoring. The vast majority of all problems attempted resulted in students in the university subject successfully completing their task without assistance. In the majority of the remaining cases the student made a request for help, and in over 25% of these requests the student then solved the problem before a tutor responded. Less than 0.5% of help requests ended in the student failing to complete the problem. Furthermore, in the case where a tutor initiated a chat thread with a struggling student, less than 4% of those attempts ended in failure.

In the school course a larger proportion of the problems (98.5%) were solved by students without help from a tutor. This may be due to the intentional accessibility of the problems. Additionally, since the course is often attempted during school class hours, students may rely more on teachers and peer support. A larger proportion of submissions also benefited from tutors reaching out to struggling students who had not asked for help themselves, due to the process of regularly reviewing this group in the triage page. Possibly because of this, the number of students failing without help was substantially lower in the school course than in the university subject.

4.3.1 Discrete-time Markov modelling. In order to understand the attrition in numbers of students attempting questions, the sequence of attempts at consecutive problems by a student can be modelled as a discrete-time Markov process, with the outcome for each problem restricted to “Pass” when all tests are completely passed before the question deadline, “Fail” when the problem is attempted but not passed before the deadline, “Assist” when the problem was passed after a response from a tutor in the live tutor chat, or “Skip” if no
attempt was made at the problem. The transition probabilities for each case are shown in Figure 10.

For the university subject (Figure 10a) problems in each worksheet were treated as distinct Markov chains to exclude the jumps in attempts for the first problem in each worksheet. The probabilities of repeating the same outcome in the next problem are high for each state. The exception to this is a skip pass, which tends to lead to a subsequent pass, suggesting students don’t generally become dependent upon tutors. Most branching occurs after a failure, where the chance of a student attempting and passing the next problem, or skipping the next problem entirely, are of similar probabilities. This illustrates the importance of tutor interactions in steering students away from failures, which then often lead to students abandoning attempts at further problems.

The school course problems for each week were also grouped as distinct chains, as for the university subject worksheets. The transition probabilities for each stream were broadly similar for each stream except for the advanced stream, very likely because of the intentional difficulty of the problems in that stream. Figure 10b groups all streams together, except for the advanced stream, which is presented separately in Figure 10c.

The non-advanced stream’s behaviour can be seen to be very similar to that in the university subject, with the key difference that students are much less likely to recover from failure, either failing the next problem or skipping it completely. This may reflect lower resilience to failure in these young learners, or the lower expectations placed on the students to complete the whole course. Again, tutor assistance can be seen to play an important role in keeping students on a successful path, but there appears to be more dependence (24% repeat usage) compared to the university subject (13%).

For the advanced school course stream, students were more likely to skip questions after receiving assistance from tutors or failing a problem, compared to the other streams. This may reflect a decision by students that this stream is beyond their abilities.

5 CONCLUSION

Online tutoring provides a useful means of supporting students in online programming learning activities, allowing students within large cohorts to communicate with tutors on individual issues. The case studies have been presented here of online tutoring used in very different contexts. In a first-year introductory programming course, tutors were able to support students at a ratio of one student to every fifty students (in Semester 1), in an efficient manner by providing scheduled periods during which tutors were guaranteed to be available. Despite the limited person-hours directed towards supporting such a large cohort, and with no weekend availability, almost half of all students made use of this facility at least once during the course.

In a large-scale school programming course, the online tutoring provides a valuable means for young students to access support beyond that available from teachers during classroom hours. Usage within this cohort was lower than for the university course, and appears to be better suited to older students comfortable with the format of holding a typed dialogue with a stranger. In such a setting it is important that tutors have suitable training, especially as students can become dependent upon the tutor and require protracted dialogue to reach a successful outcome, without necessarily developing the intended learning outcomes.

In both contexts, students are observed to attempt fewer of the later questions in each week’s content, and the number of students attempting any questions declines over time. Discrete time Markov chain modelling of the outcomes suggests that students are likely to abandon attempting later problems after failing to complete an earlier one. Interactions with a tutor can act to reduce this attrition, giving students an extra chance to move back into the stable passing track. Anecdotally from the authors’ own experiences the specific help required often amounts to correcting a small misunderstanding or pointing out a minor syntactic mistake, requiring little time for a trained tutor to identify and explain to the student. This is borne out by the fact that many message threads are very short, most often a single exchange between the tutor and the student. While there are many mechanisms by which these interactions can be achieved, the online tutoring system used in the cases described here provides important features, such as individualised message threads and access to a runnable copy of the student’s code, that help the tutor to resolve problems quickly. As such, this approach is well suited to supporting students efficiently in large programming classes.

5.1 Lessons for practitioners

A coordinator considering using online tutoring in their own course offering may find the following lessons useful, drawing on the experiences described above.

(1) Online tutoring does not need to increase the teaching budget, if some face-to-face tutoring is traded for scheduled monitoring of the online support requests for defined periods. Importantly, real-time responses are not necessary, and students may benefit from the time in a delay encouraging them to reflect and find their own solution.

(2) To improve the utility of the online tutoring, students should be reminded of its availability, particularly as they may only have use of it late in the course. Students can also be encouraged to perform the operation of starting a chat thread in order to familiarise them with the process. For instance, during outreach workshops supporting the school course, groups of high-school students were encouraged to greet the tutors via this interface (and extra tutors were on hand to return the greeting).

(3) Demand for assistance is often predictable, particularly where data is available from past offerings, and scheduling can be optimised to ensure tutors are available when they are most needed. In particular, load is, inevitably, highest close to deadlines, so extra staff could be rostered to spread the load. It is also useful to have a means for on-duty tutors to alert other tutors when a large backlog of requests occurs.

(4) Tutors should have strategies to avoid students developing a dependency for help, as observed in some cases within Figure 5. These might include delaying responses, particularly after dealing with the original issue, to encourage some independence. Students could also be required to perform definite steps, such as running code with suggested
changes, to prevent the conversation simply becoming a private lesson. Where available, struggling students could also be directed to complete simpler introductory exercises before returning to the original problem.

6 FUTURE WORK

Within the courses described in Section 3, some changes will be considered for future offerings. In the school course the observed behaviour of a small number of students to become overly dependent upon tutor support appears to be due to those students having profound and continued misconceptions but who are also extremely persistent. These students may need to be flagged as requiring special treatment, so that tutors are forewarned when the same student requests help in subsequent questions. At the other end of the scale, the current criteria for struggling students currently appears to leave a significant number of students unattended, and who give up prematurely. More effective criteria will be considered in future.

Survey responses, outside the scope of what can be presented here, suggest that there are multiple reasons why school students don’t talk to the tutors, including not knowing if they are allowed to, or not wanting to talk to them because they are strangers. In the 2019 offering of the course, students were provided with more guidance on the tutor support within the introductory slides and problems. Future analysis will show whether this change has resulted in an increase in the number of students who feel able to contact the tutors for help, and whether there has been a corresponding reduction in the attrition rate of active students as the course progresses. Changes to the platform to provide classroom teachers with visibility of tutor threads for their students may alleviate students’ wariness in communicating with tutors.

In the university course, further improvements lie in making it easier to provide more specific feedback to students, for instance by allowing tutors to highlight relevant sections of code and make these selections visible to the student. Additionally, review of message threads and analysis of the use of saved responses is likely to reveal common areas of difficulty that can be used to improve the content and structure of future offerings.

Beyond the courses described here, it is likely that the online tutoring approach would have utility in other use cases. It would be interesting to observe the efficacy of using online tutoring to support smaller classes, such as in school classrooms, particularly in rural and distance education settings, or in peer-tutoring environments.

REFERENCES