On Generating Interleaved Spaced Practice Schedules

Kevin K. H. Cheung
kevin.cheung@carleton.ca
School of Mathematics and Statistics
Carleton University
1125 Colonel By Dr.,
Ottawa, ON K1S 5B6
Canada

Abstract

A number of studies have shown the effectiveness of interleaved spaced practice for learning mathematics. Scheduling interleaved spaced practice is manually feasible when the number of skills involved is low. When the number of skills is large, designing interleaved spaced practice schedule manually can become prohibitive. In this note, an approach for automatic generation of interleaved spaced practice schedules using techniques from operations research is described. An accompanying proof-of-concept Linux/Unix command-line tool is included. The tool is open source and is made freely available to spur further research interest in interleaved spaced practice and the design of competency-based learning systems.

1 Introduction

Learning mathematics requires mastery of many skills as well as retention of skills learned. Skill retention is critical to learning mathematics because new skills often depend on old skills. Given the ever-growing body of mathematical knowledge and the increase in the use of sophisticated mathematical tools in modern applications, mathematics educators are faced with the challenge of helping students learn as quickly as possible and retain what they have learned as much and as long as possible. Poor retention leads to having to spend time on relearning or remediation.

As it is said, "Practice makes perfect." An obvious way to attain proficiency and to remain so is to practice frequently. However, as the number of acquired skills grow, practicing every skill that one has acquired frequently becomes unrealistic. (For instance, Khan Academy [1] lists over one thousand skills in its World of Math.) In addition, there is no need to practice every skill at the same frequency. Time is better spent on practicing less proficient skills than proficient ones.

A popular approach to practicing mathematical skills as evidenced by the vast majority of mathematics textbooks is massed (or blocked) practice. Massed practice involves (over-)practicing a new concept or skill, soon after it is learned, a high number of exercises within a concentrated time period.

Once the exercises for a particular concept are completed, one moves on to the next topic and the process repeats.

Another approach is spaced practice. The basic idea of spaced practice is to spread the practice over time rather than in concentrated periods. This approach has been used in language programs such as Pimsleur [3] and Duo Lingo [4]. There are even mobile apps, such as Anki [5], that implement spaced practice for more general skill acquisition.

As learning mathematics involves multiple topics, spaced practice in mathematics also often involves interleaving as found in Saxon Math Curriculum [2]. Interleaving involves practicing different skills in one particular session whereas pure spaced practice involves only one particular skill. Exercises for a particular topic are spaced throughout a textbook or even across different grade levels. In other words, in the exercise section of a particular topic, one also finds questions on topics covered in earlier sections. This approach is also called the spiral approach. A number of studies have shown the effectiveness of interleaved practice in improving learning mathematics. (See, for example, [6–9]).

Coming up with a interleaved spaced practice schedule manually does not seem challenging when the number of topics is small and the topics follow the same spacing scheme. However, in reality, a course or a textbook often covers a dozen or more topics, some of which contain dependencies on other topics. In addition, not all topics require the same amount of work to learn initially or contain similar number of practice exercises. As a result, designing a practice schedule that respects the interleaving and spacing constraints with a balanced workload across practice sessions is best carried out by an automated process.

In this note, we describe a proof-of-concept interleaved spaced practice schedule generation approach based on techniques in operations research. In particular, we specify an integer programming model that seeks to minimize the number of practice sessions to cover all the topics subject to a specified spacing scheme and maximum workload per practice session. With further refinement, the approach can be incorporated into adaptive systems that tailor training for individual students. As part of this note, we provide a Linux/Unix command-line tool written in C++ for generating the model in CPLEX LP format (see [10]) that can be solved using appropriate software or online at the NEOS Server for Optimization [11]. It is hoped that the approach discussed in this note and the accompanying command-line tool will be useful for researchers studying spaced practice and further stimulate research in the design of competency-based learning systems.

2 An illustrative example

In this section, we illustrate the various kinds of practice schedules with a simple example.

Suppose that we want to distribute the problems for the following three topics over a number of practice sessions which will be called "assignments", each of which having no more than five problems.

1. area of a rectangle: 10 problems
2. area of a triangle: 10 problems
3. area of a circle: 10 problems

The following is a massed practice schedule without interleaving:

Assignment	Rectangle	Triangle	Circle
1	5		
2	5		
3		5	
4		5	
5			5
6			5

The following is a spaced practice schedule without interleaving:

Assignment	Rectangle	Triangle	Circle
1	5		
2		5	
3			5
4	5		
5		5	
6			5

The following is a massed practice schedule with a bit of interleaving:

Assignment	Rectangle	Triangle	Circle
1	5		
2	3	2	
3	2	3	
4		3	2
5		2	3
6			5

To form a meaningful interleaved spaced practice schedule, we specify a number of criteria. We require that each topic be practiced at least four times, with the first practice containing at least four problems. We also require that no more than one assignment be sandwiched between any two practices for each topic. In other words, if assignment 2 contains practice on the area of a rectangle, then there must be another practice on the area of a rectangle no later than assignment 4. The following is an interleaved spaced practice schedule that respects all the mentioned criteria:

Assignment	Rectangle	Triangle	Circle
1	5		
2	1	4	
3		1	4
4	1	2	2
5	2	1	2
6	1	2	2

Even for this small example, coming up with the above practice schedule by hand is not entirely trivial. As the number of topics and the complexity of the criteria increase, an automated approach is needed. In the next section, we describe an integer programming model for practice schedule generation.

3 An integer programming model

To facilitate discussion, we introduce some notation for the various pieces of information that we need to work with.

The topics are denoted by $T_1, \dots, T_k$ where k is a positive integer. Topic dependencies are specified as a set $\mathcal{D}$ of ordered pairs such that $(i, j) \in \mathcal{D}$ if and only if T_i is dependent on T_j; that is, T_j must be practiced at least once before any practice for T_i can take place. For example, if the topics are T_1, T_2, and T_3, and $\mathcal{D} = \{(3, 1), (3, 2)\}$, then T_3 is dependent on both T_1 and T_2.

The integer M denotes the maximum number of assignments one can have in the schedule. Note that if M is not sufficiently large, it might not be possible to schedule all the practices.

We use A to denote a sequence (of positive integers) of length M whose ith element A_i gives the maximum number of problems the ith assignment can have.

For each $i \in \{1, \dots, k\}$, we have

- the positive integer N_i denoting the total number of practice problems for topic T_i;
- the sequence $P_i = ((p_{i1}, q_{i1}), (p_{i2}, q_{i2}), \dots, (p_{ir_i}, q_{ir_i}))$ where r_i is a positive integer at most M and $1 \leq p_{ij} \leq q_{ij} \leq N_i$ for all $j = 1, \dots, r_i$. The values p_{ij} and q_{ij} specify the lower bound and upper bound on the number of problems from topic T_i appearing in the jth practice, respectively. We will make the assumption that $N_i \geq \sum_{j=1}^{r_i} p_{ij}$ because no schedule exists otherwise;
- the positive integer K_i denoting maximum number of problems for topic T_i that can appear in each practice beyond the r_ith practice. To see why this number could be useful, note that the integer r_i (the number of elements in the sequence P_i) should be thought of as the *minimum* number of practices for topic T_i that must take place. In reality, additional practices should be allowed because we might not use up all the problems for topic T_i in the first r_i practices. If we do not want to impose such a restriction, we can simply set $K_i = N_i$.
- the sequence $S_i = ((l_{i1}, u_{i1}), (l_{i2}, u_{i2}), \dots, (l_{is_i}, u_{is_i}))$ where s_i is a positive integer at most $M - 1$ and $0 \leq l_{ij} \leq u_{ij} \leq M - 1$ for all $j = 1, \dots, s_i$. The values l_{ij} and u_{ij} specify the lower bound and upper bound on the number of assignments between the jth practice and the $(j + 1)$th practice of topic T_i, respectively. Note that $l_{ij} = 0$ means that the $(j + 1)$th practice for topic T_i can be scheduled right after the jth practice. We will use the convention that the jth element of S_i, when $j > s_i$, be $(0, \infty)$. In other words, we can think of S_i as a sequence of length $M - 1$ with only the first s_i elements to be potentially different from $(0, \infty)$. With this convention, we do not need to specify S_i if we simply need the $(j + 1)$th practice of topic T_i to be scheduled after the jth practice for all j.

Hence, for the example in Section 2, we have $M = 6$, $A = (5, 5, 5, 5, 5, 5)$, and $k = 3$. Note that it does not really matter which topic T_i represents since the data are the same for all three topics and there are no topic dependencies in the example. Thus, $N_1 = N_2 = N_3 = 10$, $P_1 = P_2 = P_3 = ((4, 10), (1, 10), (1, 10), (1, 10))$, $S_1 = S_2 = S_3 = ((0, 1), (0, 1), (0, 1), (0, 1), (0, 1))$.

With all required the notation in place, we now describe our integer programming model. First of all, the variables for our integer programming model are listed in Table 1.

Variable	Value
$x_{ai} \in \mathbb{Z}_+$ $a = 1, \dots, M,$ $i = 1, \dots, k$	the number of problems from topic T_i assigned to assignment a
$y_a \in \{0, 1\}$ $a = 1, \dots, M$	equals 1 if and only if assignment a has at least one problem
$n_{ij} \in \mathbb{Z}_+$ $i = 1, \dots, k,$ $j = 1, \dots, M$	the number of problems from topic T_i allocated towards the jth practice
$w_{ij} \in \{0, 1\}$ $i = 1, \dots, k,$ $j = r_i + 1, \dots, M$	equals 1 if and only if there is a jth practice for topic T_i
$z_{ij} \in \{0, 1, \dots, M\}$ $i = 1, \dots, k,$ $j = 1, \dots, M$	the number of the assignment in which the jth practice of topic T_i takes place
$s_{ija} \in \{0, 1\}$ $i = 1, \dots, k,$ $j = 1, \dots, M$ $a = 1, \dots, M$	equals 1 if and only if the jth practice of topic T_i takes place in assignment a.

Table 1: Variables

Next, describe the constraints relating the variables.

For each $a = 1, \dots, M$, y_a needs to equal 1 when $x_{ai} > 0$ for some i. We can enforce this and the upper bound on the number of problems on assignment a by the constraint

$$\sum_{i=1}^{k} x_{ai} \leq A_a y_a.$$

For each $i = 1, \dots, k$, we have constraints described below.

1. If all the problems for topic T_i must be scheduled, we will need the constraint

$$\sum_{j=1}^{M} n_{ij} = N_i.$$

Otherwise, we will have the constraint

$$\sum_{j=1}^{M} n_{ij} \leq N_i.$$

2. For $j = 1, \dots, r_i$, we have the bound constraints

$$p_{ij} \leq n_{ij} \leq q_{ij}$$

and for $j = r_{i+1}, \dots, M$, the constraint

$$n_{ij} \leq K_i.$$

3. For $j = r_i + 1, \ldots, M$, $w_{ij} = 1$ if and only if $n_{ij} > 0$. As n_{ij} cannot exceed N_i, the condition can be enforced by the constraints

$$n_{ij} \leq N_i w_{ij} \leq N_i n_{ij}.$$

   To ensure that there is no practice of zero problem followed by a practice with a positive number of problems, we impose the constraints

$$w_{ij} \leq w_{ij-1} \quad j = r_i + 2, \ldots, M.$$

4. The jth practice must be assigned to exactly one assignment for each $j = 1, \ldots, r_i$. As we may not necessarily need to have more than r_i practices, we enforce that, for $j > r_i$, the jth practice be assigned to an assignment if and only if there will be a jth practice. Thus, for each $i = 1, \ldots, k$, we have the constraints:

$$\begin{aligned} \sum_{a=1}^{M} s_{ija} &= 1 \quad j = 1, \ldots, r_i, \\ \sum_{a=1}^{M} s_{ija} &= w_{ij} \quad j = r_i + 1, \ldots, M. \end{aligned}$$

   Then, the assignment number z_{ij} for the jth practice can be set using the constraint

$$z_{ij} = \sum_{a=1}^{M} a s_{ija}$$

   for $j = 1, \ldots, M$.

5. If $s_{ija} = 0$ for all $j = 1, \ldots, M$, we need x_{ai} to be 0 because no practice of topic T_i is assigned to assignment a. This is enforced with the constraint

$$x_{ai} \leq N_i \sum_{j=1}^{M} s_{ija}.$$

   However, if $s_{ija} = 1$ for some $j \in \{1, \ldots, M\}$ (that is, the jth practice is assigned to assignment a), we need x_{ai} to equal n_{ij}. This can be enforced with the constraints

$$n_{ij} - N_i(1 - s_{ija}) \leq x_{ai} \leq n_{ij} + N_i(1 - s_{ija})$$

   Note that when $s_{ija} = 1$, the constraint reduces to

$$x_{ai} = n_{ij}.$$

   However, when $s_{ija} = 0$, the constraint reduces to

$$x_{ai} \leq n_{ij} + N_i$$

   which holds trivially since x_{ai} cannot exceed the total number of problems for topic T_i.

6. The spacing of successive practices are enforced by the constraints

$$l_{ij-1} + 1 \leq z_{ij} - z_{ij-1} \leq u_{ij-1} + 1 \quad j = 2, \ldots, r_i$$

for the first r_i practices, and by the constraints

$$(l_{ij-1} + 1 + M)w_{ij} - M \leq z_{ij} - z_{ij-1} \leq (u_{ij-1} + 1 - M)w_{ij} + M \quad j = r_i + 1, \ldots, M$$

for the remaining practices, if any. Recall that we use the convention that $l_{ij} = 0$ and $u_{ij} = \infty$ if $j > s_i$.

To enforce topic dependencies, we include the constraints

$$z_{i_1 1} \geq z_{i_2 1} + 1 \ \ \forall (i_1, i_2) \in \mathcal{D}$$

Then, to obtain the largest assignment number over all assignments with a positive number of problems, we minimize z subject to the above constraints and

$$z \geq a y_a \qquad a = 1, \ldots, M.$$

In summary, the integer programming model for the case when all the problems must be scheduled is as shown in Figure 1.

4 Command-line tool

As a proof of concept, we developed a Linux/Unix command-line tool, written in C++, that accepts from the standard input the various parameters for the model and outputs a file in CPLEX LP format containing the integer programming model to be solved. The source code and usage details can be found in the accompanying files.

We used it to generate a schedule for the following test case. Four topics that appear in Khan Academy's [1] knowledge map were picked. For each topic, the number of problems was arbitrarily set.

- T_1: adding 1's or 10's (no regrouping), with $N_1 = 16$ problems
- T_2: regrouping when adding one-digit numbers, with $N_2 = 13$ problems
- T_3: adding two-digit numbers (no regrouping), with $N_2 = 11$ problems
- T_4: adding two-digit numbers by making tens, with $N_4 = 10$ problems

The knowledge map gives the following topic dependencies: $\mathcal{D} = \{(2,1), (3,1), (4,2), (4,3)\}$. That is, T_2 and T_3 depend on T_1 and T_4 depends on both T_1 and T_2.

Suppose that there is a maximum of eight assignments, each of which can contain no more than seven problems. For each topic, we want to practice at least three times, each of which with at least three problems and at most five problems. We also want practices beyond the third one to contain no

$$
\begin{array}{lll}
\min & z & \\
\text{s.t.} & z \geq a y_a & a = 1, \ldots, M \\
& \displaystyle\sum_{i=1}^{k} x_{ai} \leq A_a y_a, & a = 1, \ldots, M \\
& z_{i_1 1} \geq z_{i_2 1} + 1 & \forall (i_1, i_2) \in \mathcal{D} \\
& \left.\begin{array}{ll}
l_{ij-1} + 1 \leq z_{ij} - z_{ij-1} \leq u_{ij-1} + 1 & j = 2, \ldots, r_i \\
z_{ij} - z_{ij-1} \leq (u_{ij-1} + 1 - M) w_{ij} + M & j = r_i + 1, \ldots, M \\
(l_{ij-1} + 1 + M) w_{ij} - M \leq z_{ij} - z_{ij-1} & j = r_i + 1, \ldots, M \\
n_{ij} - N_i (1 - s_{ija}) \leq x_{ai} \leq n_{ij} + N_i (1 - s_{ija}) & j = 1, \ldots, M, a = 1, \ldots, M \\
x_{ai} \leq N_i \displaystyle\sum_{j=1}^{M} s_{ija} & a = 1, \ldots, M \\
z_{ij} = \displaystyle\sum_{a=1}^{M} a s_{ija} & j = 1, \ldots, M \\
\displaystyle\sum_{a=1}^{M} s_{ija} = 1 & j = 1, \ldots, r_i, \\
\displaystyle\sum_{a=1}^{M} s_{ija} = w_{ij} & j = r_i + 1, \ldots, M. \\
n_{ij} \leq N_i w_{ij} \leq N_i n_{ij} & j = r_i + 1, \ldots, M. \\
w_{ij} \leq w_{ij-1} & j = r_i + 2, \ldots, M, \\
p_{ij} \leq n_{ij} \leq q_{ij} & j = 1, \ldots, r_i, \\
n_{ij} \leq K_i & j = r_{i+1}, \ldots, M, \\
\displaystyle\sum_{j=1}^{M} n_{ij} = N_i. & \\
y_a \in \{0, 1\} & a = 1, \ldots, M \\
w_{ij} \in \{0, 1\} & j = r_i + 1, \ldots, M \\
s_{ija} \in \{0, 1\} & j = 1, \ldots, M, a = 1, \ldots, M \\
x_{ai}, n_{ij}, z_{ij} \in \mathbb{Z}_+ & j = 1, \ldots, M, a = 1, \ldots, M
\end{array}\right\} & i = 1, \ldots, k
\end{array}
$$

Figure 1: Integer programming model

more than three problems. Then $P_i = ((3,5),(3,5),(3,5))$ and $K_i = 3$ for $i = 1,\ldots,4$. We also have $M = 8$ and $A = (7,7,7,7,7,7,7,7)$.

For the spacing scheme, suppose that there must at least one and at most two practices between the first three practices for each topic, and at least two and at most four practices between the subsequent practices for each topic. Then $S_i = ((1,2),(1,2),(2,4),(2,4),(2,4),(2,4),(2,4))$ for $i = 1\ldots,4$.

Solving the integer programming model with IBM ILOG CPLEX 12.5.1.0 gives the schedule shown in Table 2. A copy of the Mac OS X terminal session for obtaining this schedule is given in the appendix.

Assignment	T_1	T_2	T_3	T_4
1	5			
2		4	3	
3	4			3
4		4	3	
5	4			3
6		5		
7			5	
8	3			4

Table 2: Interleaved spaced schedule

One can readily check that the above schedule satisfies all the constraints. As the model minimizes the number of assignments subject to the given constraints, one concludes that there is no schedule having fewer than eight assignments.

5 Concluding remarks

There is supporting evidence that interleaved spaced practice can improve the learning of mathematics. We developed a proof-of-concept command-line tool for automating interleaved spaced practice schedule generation based on an integer programming model. The constraints that are taken into account capture some of the most prominent criteria for spaced schedules. The model can be extended in various ways to accommodate more refined criteria.

The intention behind developing this tool and making it freely available is to spur further research interest in interleaved spaced practice as well as on generating interleaved spaced schedules in an offline or online setting. Incidentally, the integer programming model itself is worth investigating further as solving such a model for more than a dozen topics with nontrivial dependencies could take a substantial amount of computing resources. It is conceivable that alternate formulations or solution methods could improve solution times.

Appendix

The command-line tool is named `genip` if built from the source file `genip.cpp` [S1] using the accompanying Makefile [S2]. Instructions for building and using `genip` are given in `README.txt` [S3]. All the relevant files are accessible under the Supplementary files section.

The tool accepts from the standard input a sequence of numbers following a certain format. A convenient way to use the command-line tool is to first create a text file containing all the input numbers using a text editor (e.g. nano, emacs, vi etc.). Below is the content of the file (`paper.txt` [S4] in the accompanying files) we created for the example in Section 4:

```
8   7 7 7 7 7 7 7 7
4   2 1  3 1  4 2  4 3
4
16 3  3  3 5  3 5  3 5    7  1 2  1 2  2 4  2 4  2 4  2 4  2 4
13 3  3  3 5  3 5  3 5    7  1 2  1 2  2 4  2 4  2 4  2 4  2 4
11 3  3  3 5  3 5  3 5    7  1 2  1 2  2 4  2 4  2 4  2 4  2 4
10 3  3  3 5  3 5  3 5    7  1 2  1 2  2 4  2 4  2 4  2 4  2 4
```

The first number that the tool expects is the number of assignments. It then expects the sequence of numbers A. In our example, the number of assignments is 8 and $A = (7, 7, 7, 7, 7, 7, 7, 7)$. The first line of the file shown above shows these numbers.

The tool then expects the number of dependency pairs followed by the actual pairs of numbers for the dependencies. In the second line of the file, we see the number 4 for the four topic dependencies followed by four pairs of numbers that specify the dependencies.

The tool then expects the number of topics. In our case, the number is 4 and we see this number in the fourth line of the file.

Once the number of topics k is read, the tool then expects k sequences of numbers, with the ith sequence containing the following: the number of problems for topic T_i, the number r_i, the pairs of numbers in P_i, the number s_i, and the pairs of numbers in S_i. For example, in the last line of the file, we have the sequence for topic T_4. In this sequence, the first number, 10, is the number of problems for topic T_4. The next number, 3, is the number of pairs in P_4. The three pairs of numbers that follow are the pairs in P_4. Then the number 7 is the number of pairs in S_4 and the seven pairs of numbers that follow are the pairs in S_4.

Below is a Mac OS X terminal session showing how the tool was run and how IBM ILOG CPLEX was used to obtain the schedule. (The same commands should work on Linux or Unix in a bash shell with IBM ILOG CPLEX installed. Instructions for using the online NEOS Optimization Server are given in the accompanying `README.txt` [S3] file.) Note that the first command directs the content of `paper.txt` to the standard input.

```
bash-3.2$ ./genip < paper.txt
# of assignments: 8
Number of topics: 4

# problems for topic 1: 16
Practice bounds:
(3, 5) (3, 5) (3, 5)
Spacing bounds:
(1, 2) (1, 2) (2, 4) (2, 4) (2, 4) (2, 4) (2, 4)

# problems for topic 2: 13
Practice bounds:
```

```
(3, 5) (3, 5) (3, 5)
Spacing bounds:
(1, 2) (1, 2) (2, 4) (2, 4) (2, 4) (2, 4) (2, 4)

# problems for topic 3: 11
Practice bounds:
(3, 5) (3, 5) (3, 5)
Spacing bounds:
(1, 2) (1, 2) (2, 4) (2, 4) (2, 4) (2, 4) (2, 4)

# problems for topic 4: 10
Practice bounds:
(3, 5) (3, 5) (3, 5)
Spacing bounds:
(1, 2) (1, 2) (2, 4) (2, 4) (2, 4) (2, 4) (2, 4)
bash-3.2$ cplex

Welcome to IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer 12.5.1.0
  with Simplex, Mixed Integer & Barrier Optimizers
5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
Copyright IBM Corp. 1988, 2013.  All Rights Reserved.

Type 'help' for a list of available commands.
Type 'help' followed by a command name for more
information on commands.

CPLEX> set mip display 0
New value for level of mixed integer node display: 0
CPLEX> read ip.lp
Problem 'ip.lp' read.
Read time = 0.01 sec. (0.08 ticks)
CPLEX> o

Solution pool: 1 solution saved.

MIP - Integer optimal solution:  Objective =  8.0000000000e+00
Solution time =    0.03 sec.  Iterations = 23  Nodes = 0
Deterministic time = 5.95 ticks  (217.52 ticks/sec)

CPLEX> disp sol var x*
Incumbent solution
Variable Name           Solution Value
x1_1                          5.000000
x2_2                          4.000000
x2_3                          3.000000
x3_1                          4.000000
x3_4                          3.000000
```

```
x4_2                          4.000000
x4_3                          3.000000
x5_1                          4.000000
x5_4                          3.000000
x6_2                          5.000000
x7_3                          5.000000
x8_1                          3.000000
x8_4                          4.000000
All other variables matching 'x*' are 0.
CPLEX>
```

In the CPLEX session above, the command `set mip display 0` was used to suppress messages from the mixed-integer programmming solver. It can be omitted if the messages are desired.

The command `read ip.lp` loads the model contained in the file `ip.lp` created by `genip`.

The command `opt` invokes the solver to solve the model.

The command `disp sol var x*` asks for a listing of the solution values of all the variables beginning with the letter `x`. The variable `xi_j` corresponds to the variable x_{ij} in our integer programming model. For example, `x3_4` is the number of problems from topic T_4 appearing in assignment 3.

References

[1] Khan Academy. `http://www.khanacademy.org`. Last accessed: Jan 19, 2016.

[2] Saxon math curriculum. `http://www.hmhco.com/shop/education-curriculum/math/saxon-math`. Last accessed: Jan 17, 2016.

[3] Pimsleur. `http://www.pimsleur.com`. Last accessed: Jan 17, 2016.

[4] Duo Lingo. `http://www.duolingo.com`. Last accessed: Jan 17, 2016.

[5] Anki. `http://ankisrs.net`. Last accessed: Jan 21, 2016.

[6] D. Rohrer and K. Taylor. The shuffling of mathematics practice problems boost learning. *Instructional Science*, **35**:481–498, 2007.

[7] D. Rohrer, R.F. Dedrick, K. Burgess. The benefit of interleaved mathematics practice is not limited to superficially similar kinds of problems. *Psychonomic Bulletin and Review*, **21**(5):1323–1330, 2014.

[8] D. Rohrer, R.F. Dedrick, S. Sterschic. Interleaved practice improves mathematics learning, *Journal of Educational Psychology*, **107**(3):900–908, 2015.

[9] M.A. Yazdani and E. Zebrowski. Spaced reinforcement: An effective approach to enhance the achievement in plane geometry. *Journal of Mathematical Sciences and Mathematics Education*, **1**(1):37–43.

[10] CPLEX LP format. `http://plato.asu.edu/cplex_lp.pdf`. Last accessed: Jan 17, 2016.

[11] IBM ILOG CPLEX MILP solver at NEOS Server for Optimization. `http://www.neos-server.org/neos/solvers/milp:CPLEX/LP.html`. Last accessed: Jan 17, 2016.

Supplementary files

[S1] genip.cpp, C++ source file for the command-line tool generating the integer-programming model.

[S2] Makefile, makefile for building `genip`.

[S3] README.txt, instructions for building `genip` and using IBM ILOG CPLEX or the online NEOS Optimization Server to solve the integer programming model.

[S4] paper.txt, text file containing the numbers for inputting to `genip` for generating the integer-programming model for the example in Section 4.

[S5] command.txt, text file containing the commands for using the online NEOS Optimization Server.

[S6] in.txt, text file containing the numbers for inputting to `genip` for generating the integer-programming model for a small example described in README.txt.

[S7] small.txt, text file containing the numbers for inputting to `genip` for generating the integer-programming model for the example in Section 2.