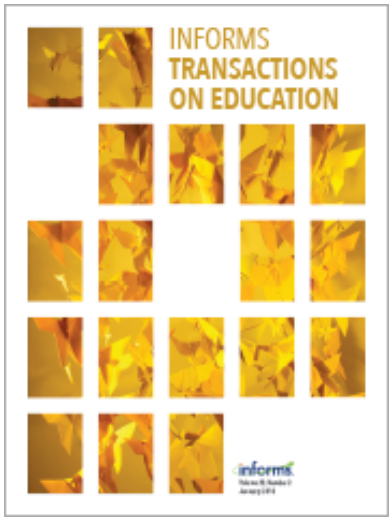
This article was downloaded by: [216.73.216.221] On: 10 November 2026, At: 15:47
Publisher: Institute for Operations Research and the Management Sciences (INFORMS)
INFORMS is located in Maryland, USA



INFORMS Transactions on Education

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Solving the n-Fractions Puzzle Using Integer Programming

Michael Forbes, Ruoxuan Shi

To cite this article:
Michael Forbes, Ruoxuan Shi (2026) Solving the n-Fractions Puzzle Using Integer Programming. INFORMS Transactions on Education

Published online in Articles in Advance 10 Nov 2026

. https://doi.org/10.1287/ited.2026.0233

This work is licensed under a Creative Commons Attribution 4.0 International License. You are free to copy, distribute, transmit and adapt this work, but you must attribute this work as "*INFORMS Transactions on Education*. Copyright © 2026 The Author(s). https://doi.org/10.1287/ited.2026.0233, used under a Creative Commons Attribution License: https://creativecommons.org/licenses/by/4.0/."

Copyright © 2026 The Author(s)

Please scroll down for article—it is on subsequent pages

With 12,500 members from nearly 90 countries, INFORMS is the largest international association of operations research (O.R.) and analytics professionals and students. INFORMS provides unique networking and learning opportunities for individual professionals, and organizations of all types and sizes, to better understand and use O.R. and analytics tools and methods to transform strategic visions and achieve better outcomes.
For more information on INFORMS, its publications, membership, or meetings visit http://www.informs.org

https://pubsonline.informs.org/journal/ited

INFORMS TRANSACTIONS ON EDUCATION
Articles in Advance, pp. 1–5
ISSN 1532-0545 (online)

Solving the *n*-Fractions Puzzle Using Integer Programming

Michael Forbes,[a] **Ruoxuan Shi**[a,*]

[a]School of Mathematics and Physics, The University of Queensland, St. Lucia, Queensland 4072, Australia

*Corresponding author

m.forbes@uq.edu.au (M. Forbes), ruoxuan.shi@student.uq.edu.au (R. Shi)

https://orcid.org/0000-0002-2240-7245 (M. Forbes), https://orcid.org/0009-0002-8392-8084 (R. Shi)

Received: June 14, 2026
Revised: July 21, 2026
Accepted: July 31, 2026
Published Online in Articles in Advance: November 10, 2026

https://doi.org/10.1287/ited.2026.0233

Copyright: © 2026 The Author(s)

Open Access Statement: This work is licensed under a Creative Commons Attribution 4.0 International License. You are free to copy, distribute, transmit and adapt this work, but you must attribute this work as "*INFORMS Transactions on Education*. Copyright © 2026 The Author(s). https://doi.org/10.1287/ited.2026.0233, used under a Creative Commons Attribution License: https://creativecommons.org/licenses/by/4.0/."

Abstract. We present a teaching case study in which we solve the *n*-fractions puzzle using a Mixed Integer Programming model. The *n*-fractions puzzle is problem 41 in CSPLib, a library of test problems for Constraint Programming solvers. We first propose a model which is correct assuming exact arithmetic and show how to modify the model to account for floating point rounding errors. We then reuse ideas from constraint programming approaches to the puzzle and limit the lowest common multiple (LCM) of the denominators in the solution. This massively reduces solution time and means we can finally close the remaining open case to the puzzle. The apparent simplicity of the puzzle, the novelty of the modelling techniques used and the speedups achieved through limiting the LCM make the puzzle an interesting teaching case study.

Funding: This work was supported by the Australian Government (RTP Scholarship).

Keywords: puzzles • integer programming

Downloaded from informs.org by [216.73.216.221] on 10 November 2026, at 15:47 . For personal use only, all rights reserved.

1. Introduction

Teaching advanced integer programming modelling and solution techniques to intermediate and advanced undergraduate students requires presenting a wide range of models and introducing a wide range of modelling techniques. One engaging way to do this is to use puzzles. In this paper we consider the *n*-fractions puzzle and show how it can be used to introduce several interesting and useful modelling techniques.

The original 3-fractions puzzle is specified as: find nine distinct nonzero digits that satisfy

$$\frac{A}{10B+C}+\frac{D}{10E+F}+\frac{G}{10H+I}=1. \tag{1}$$

This generalises to: find $3n$ nonzero digits satisfying

$$\sum_{i=1}^{n}\frac{a_i}{10b_i+c_i}=1, \tag{2}$$

where the number of occurrences of each digit is between 1 and $\lceil n/3\rceil$. We are interested in finding the greatest n for which a solution exists.

The *n*-fractions puzzle was presented as a puzzle in this journal (Malapert and Provillard 2018) and modelled as a constraint satisfactions problem (CSP). A CSP consists of a set of variables defined by their domains and a set of constraints. A constraint is a relation between a subset of variables that restricts the possible values that variables can simultaneously take. CSPs are usually solved by Constraint Programming (CP) engines such as Google's OR-Tools (Google 2024), but can often be reframed as Mixed Integer Programming (MIP) problems and solved with a specialist MIP solver such as Gurobi (2024). The choice of which solver to use is often not clear and can be dependent on the modeller's ability to express the problem in the best possible way for the environment in question.

The first appearance of the 3-fractions puzzle in the constraint satisfaction literature appears to be in Schulte and Smolka (2004). The *n*-fractions puzzle (Frisch et al. 2018) is problem 41 of CSPLib (Jefferson et al. 1999), a library of CSPs.

To the best of our knowledge, all the models proposed to date use CP to solve this puzzle. In Codish (2018) and Malapert and Provillard (2018) Equation (2) is restated as

$$\sum_{i=1}^{n}\frac{a_i\times L}{10b_i+c_i}=L, \tag{3}$$

Downloaded from informs.org by [216.73.216.221] on 10 November 2026, at 15:47 . For personal use only, all rights reserved.

where L is the lowest common multiple (LCM) of $10b_i + c_i (i = 1, \ldots, n)$. The CP model is solved with gradually increasing limits on L. The intuition here is that it is easier to hit a target when there are fewer options.

Malapert and Provillard (2018) prove that the n-fractions puzzle has no solutions for $n \geq 45$. CSPLib lists solutions for all n from 3 to 43. Many of these are from Codish (2018) and Malapert and Provillard (2018) which report run times in excess of 100 hours for the larger instances.

This paper presents the puzzle as a flexible teaching case. The teaching case aims to guide students through a rich exploration of model reformulation in the classroom. Specifically, we present an integer programming model for the puzzle, and show how to modify the solution procedure to handle the limitations of floating point arithmetic. We then incorporate bounds on the LCM into the model, producing solutions for all n from 3 to 44 with solution times mostly under 1 second and a maximum of just 4 seconds. Further, we extend the case beyond simply solving problems by using LCM as the objective function, and find solutions with the smallest possible LCM for each n, as well as verifying that many solutions exist for all n from 4 to 44. The reformulation framework also provides students with a good exercise in approaching a single problem from multiple mathematical modelling perspectives.

The novel modelling techniques used to represent the fractions, cut off approximate solutions and calculate the LCM make this an ideal teaching example for intermediate to advanced students.

This short paper is organized as follows. In Section 2 we introduce the initial model and the LCM model. In Section 3 we describe our experience with using this puzzle as a teaching example. In Section 4 we detail our computation results before ending with a few concluding remarks.

2. The Integer Programming Model

2.1. Initial Model

Let $D = \{1, \ldots, 9\}$ and $\mathscr{D} = D \times D \times D$. We define integer variables $x_{ijk}, (i,j,k) \in \mathscr{D}$ which represent the number of times $\frac{i}{10j+k}$ appears in (2). We additionally define $\delta_{t,i}$ as a function that counts the number of times i appears in the tuple t. For example, we write $\delta_{(i,j,k),\alpha}$ to count how many of i,j,k are equal to α and $\delta_{(j),\alpha}$ which is 1 if $j = \alpha$.

We then formulate our MIP, a feasibility problem, as:

$$\sum_{(i,j,k)\in\mathscr{D}} \frac{i}{10j+k} x_{ijk} = 1 \tag{4}$$

$$\sum_{(i,j,k)\in\mathscr{D}} x_{ijk} = n \tag{5}$$

$$\sum_{(i,j,k)\in\mathscr{D}} \delta_{(i,j,k),\alpha} x_{ijk} \geq 1 \qquad \forall \alpha \in D \tag{6}$$

$$\sum_{(i,j,k)\in\mathscr{D}} \delta_{(i,j,k),\alpha} x_{ijk} \leq \lceil n/3 \rceil \qquad \forall \alpha \in D \tag{7}$$

$$x_{ijk} \geq 0, integer \qquad \forall (i,j,k) \in \mathscr{D} \tag{8}$$

These directly encode the constraints of the problem.

This model quickly solves the n-fractions puzzle for small n. Additionally, it quickly returns infeasible for $n \geq 45$. However, if we use exact arithmetic to check (4), then for values of $n \geq 10$ most of the solutions found do not satisfy the constraint. This is because the MIP solver uses floating point approximations for rational valued coefficients leading to the accumulation of small rounding errors.

This issue can be corrected by additional cuts, implemented as lazy constraints. Whenever we find a candidate solution x^* we verify constraint (4) using exact arithmetic, with python Fractions library (Python Software Foundation 2024). If the constraint does not hold exactly, we add the additional constraint

$$\sum_{(i,j,k)\in\mathscr{D} \mid x^*_{ijk}=0} x_{ijk} \geq 1 \tag{9}$$

which ensures that at least one currently unused fraction will be used. This cut may possibly cut off some solutions that use the same fractions, but with different values of x_{ijk}. However, this case is likely to be very rare and has no material impact on our results.

With this constraint added we can find solutions for n up to 12 in a few seconds. However, a better approach is required to scale the model to larger n.

2.2. The LCM Model

Following the example of Codish (2018), we consider restricting the LCM of the fractions used. To do this, we need to calculate the LCM using only linear constraints. Let P be the set of prime numbers smaller than 100. We limit the LCM of any n-fraction to no more than some limit L^* with the constraint

$$\prod_{p\in P} p^{y_p} \leq L^*, \tag{10}$$

where y_p is the maximum power of p in the prime factors of the denominators of the fractions used. This can be linearised by taking the logarithm of both sides.

Additionally, we can introduce binary variables $z_{jk}, j \in D, k \in D$ which are 1 if $10j + k$ appears in the denominator of any fraction used in a solution and define γ_{pjk} to be the power of p in the prime factorisation of $10j + k$. In addition to the original model (4) to (8), the constraints required to limit the LCM are:

$$\sum_{i\in D} x_{ijk} \leq \left\lceil \frac{n}{3(1+\delta_{(j),k})} \right\rceil z_{jk} \qquad \forall j \in D, k \in D \tag{11}$$

$$y_p \geq \gamma_{pjk} z_{jk} \qquad \forall j \in D, k \in D, p \in P \tag{12}$$

$$\sum_{p\in P} log(p) y_p \leq log(L^*). \tag{13}$$

Downloaded from informs.org by [216.73.216.221] on 10 November 2026, at 15:47 . For personal use only, all rights reserved.

Constraints (11) force z_{jk} to be 1 if the denominator $10j + k$ is used. Constraints (12) set the values of y_p to the maximum power of p in any denominator that is used. Finally (13) limits the LCM. We initially set $L^* = 1000$. If no solution is obtained for a given value of L^*, we multiply it by 10 and try again until a solution is found.

3. Classroom Experience

3.1. Context

The n-fractions puzzle is used by the authors for a two-hour tutorial class in an advanced integer programming course. The class size is usually around 30 to 40 students with a mix of third- and fourth-year undergraduate students and masters students from various programs including a Bachelor of Mathematics and a Masters of Data Analytics. The advanced integer programming course is heavily advertised in an introductory course, so the students have chosen this course in the expectation that they will encounter challenging models. By the time the n-fractions puzzle is introduced they have already done some work with logic-based Benders decomposition and models with composite variables—variables which represent several underlying decisions. Students all have laptops with Gurobi installed. As the required code is relatively short, they can code along and try their own ideas.

Malapert and Provillard (2018) also report their teaching experience with the n-fractions puzzle. Their context shared some similarities in the level of students and size of the class, but was also different as they were teaching computer science students. They introduced the problem in one session, the students did some modelling work as homework and perhaps encountered floating point errors with (4). In the next session these problems were explored in more detail and LCM based approaches introduced.

Clearly the appeal of the puzzle applies in different contexts. It is simple to explain, rich in detail, with the possibility of many eye-opening moments for students.

3.2. Challenges

In our experience, the first difficulty students face is identifying the correct variables. A common initial idea is to have variables to determine the digits used in each position in the ith fraction. At this stage we point out that such a formulation is very symmetrical and, more importantly, that it is not possible to express constraint (2) linearly. This usually results in some students correctly identifying the x_{ijk} variables used in this paper. A common modelling inefficiency here is to make the variables binary with another index for each of the n fractions. Repeating the emphasis on symmetry helps students understand that integer variables can be used.

Once the problem is solved for small values of n we get the students to use a suitable exact arithmetic tool to calculate the left-hand side of (2). For some this is the first exposure to floating point approximation issues. Students often try to tighten the floating point tolerances of Gurobi, but quickly realise something more is required. Their experience with logic-based Benders decomposition means some identify the need to use lazy constraints.

At this stage the problem is still difficult to solve, so we introduce the notion of a restricted LCM range from Codish (2018). It is still far from obvious how to calculate the LCM using linear constraints. Through a series of questions, we obtain a definition of the LCM corresponding to the left-hand side of (10). Due to the relative maturity of the students, this is often enough.

When teaching a new problem in class, the authors find it useful to get the students to explore similar problems, with slightly modified constraints or objectives. In this case, the students made several suggestions that were explored by the students in small groups. These suggestions and the results of the calculations are discussed in the computational results.

3.3. Student Response

This tutorial is regularly one of the most animated of the course. The students who self-select for this course enjoy puzzles and enjoy having to combine two techniques that they have previously used: composite variables and logic-based Benders decomposition.

Additionally, they are exposed to the new idea of restricting the search space, or using an objective as described in our Computational Results, in an attempt to find feasible solutions to a difficult integer programming problem. The authors provide examples from other domains where this may be useful—for example, a vehicle routing problem where the primary concern is to find a feasible solution with a fixed number of vehicles and minimising the total distance travelled helps the search process.

4. Computational Results

The models were implemented in Python and solved using Gurobi 11.0 on an x64 based PC equipped with 64 GB of RAM and an Intel i9 processor running at 2500 MHz. This processor has 14 cores and 20 logical processors but was limited to 8 threads when running the model. We set Gurobi's branching direction parameter (BranchDir) to 1, as preliminary tests indicated that this helped find solutions. All runs were restricted to a run time limit of five minutes.

In our first test we ran the initial model, without lazy constraints. This produced solutions quickly, but when checked with exact arithmetic the solutions were invalid except for $n \in \{3,4,5,6,7,9\}$. The model also returns infeasible, in around 0.01 seconds, for $n \geq 45$.

Downloaded from informs.org by [216.73.216.221] on 10 November 2026, at 15:47 . For personal use only, all rights reserved.

We next ran the initial model with lazy constraints for $n \le 20$. For $n \le 12$ solutions were found in a few seconds, and for $n = 15$ a solution was found in 20 seconds. For other n no solution was found in the time limit, though several hundred approximate solutions were cut off by lazy constraints and several million branch and bound nodes were explored.

The LCM model produced outstanding results. Solutions were found for all $3 \le n \le 44$, with the largest run time being around 4 seconds for $n = 44$. The LCM bound needed to increase from 1,000 to 10,000 for $n \ge 33$ and all the way to 1,000,000 for $n = 44$.

We separately modified the model to remove constraint (13) and instead use the left-hand side of this equation as the objective in a minimisation problem. This gives the minimum LCM solution for each n. Table 1 lists n, the run times in seconds T, the number of constraints (9) added to cut off solutions that did not satisfy constraint (4) when calculated with exact arithmetic, the best LCM and the fractions in the solution found in each case. It is striking how small most LCMs are and how few cuts were required due to low values of the LCM.

To explore the number of solutions we modified the call-back to store any solution found and add a constraint of the form (9) to cut off the solution even though it has been verified with exact arithmetic. In this way we can search for many solutions. For $n > 3$ there appear to be very many solutions. Even for $n = 44$ this procedure found 16 different solutions in five minutes, while also eliminating 1,969 approximate solutions and searching 1.6 million branch and bound nodes. Solutions are different in the sense that each solution uses at least one fraction that does not appear in any of the other solutions.

Table 1. Solutions with Smallest LCM

n	T	Cuts	LCM	Fractions
3	0.27	0	204	$\frac{5}{34}+\frac{7}{68}+\frac{9}{12}$
4	0.71	0	54	$\frac{3}{54}+2\cdot\frac{6}{27}+\frac{9}{18}$
5	0.43	0	54	$\frac{3}{27}+\frac{3}{54}+\frac{6}{27}+\frac{8}{18}+\frac{9}{54}$
6	1.16	0	72	$\frac{4}{18}+\frac{4}{36}+\frac{5}{18}+\frac{5}{36}+2\cdot\frac{9}{72}$
7	0.72	1	54	$\frac{1}{18}+3\cdot\frac{3}{27}+\frac{6}{18}+\frac{6}{54}+\frac{9}{54}$
8	0.69	0	68	$2\cdot\frac{2}{17}+\frac{2}{68}+\frac{3}{34}+\frac{5}{17}+\frac{5}{34}+\frac{5}{68}+\frac{9}{68}$
9	0.77	0	76	$\frac{2}{19}+\frac{2}{38}+\frac{2}{76}+\frac{4}{19}+2\cdot\frac{4}{38}+\frac{5}{19}+2\cdot\frac{5}{76}$
10	0.41	0	56	$2\cdot\frac{1}{14}+4\cdot\frac{3}{28}+2\cdot\frac{4}{56}+\frac{7}{56}+\frac{9}{56}$
11	0.58	0	76	$\frac{1}{19}+2\cdot\frac{2}{19}+\frac{2}{38}+\frac{2}{76}+\frac{4}{38}+\frac{4}{76}+\frac{5}{38}+2\cdot\frac{5}{76}+\frac{9}{38}$
12	0.75	0	78	$4\cdot\frac{1}{39}+3\cdot\frac{4}{26}+\frac{4}{78}+\frac{5}{26}+3\cdot\frac{5}{78}$
13	0.63	0	76	$\frac{1}{19}+3\cdot\frac{2}{19}+2\cdot\frac{2}{38}+\frac{3}{76}+2\cdot\frac{4}{38}+2\cdot\frac{4}{76}+\frac{5}{76}+\frac{8}{76}$
14	0.56	0	78	$5\cdot\frac{1}{26}+\frac{3}{39}+2\cdot\frac{4}{39}+2\cdot\frac{4}{78}+3\cdot\frac{5}{78}+\frac{9}{39}$
15	0.59	0	156	$4\cdot\frac{1}{26}+\frac{1}{39}+4\cdot\frac{4}{39}+\frac{4}{52}+4\cdot\frac{5}{78}+\frac{6}{78}$
16	0.47	0	78	$4\cdot\frac{1}{26}+2\cdot\frac{1}{78}+2\cdot\frac{2}{39}+4\cdot\frac{4}{78}+4\cdot\frac{5}{39}$
17	0.76	0	96	$4\cdot\frac{1}{32}+2\cdot\frac{1}{48}+\frac{2}{32}+\frac{3}{48}+3\cdot\frac{5}{48}+2\cdot\frac{5}{96}+4\cdot\frac{7}{96}$
18	0.83	0	288	$2\cdot\frac{1}{18}+2\cdot\frac{1}{48}+\frac{3}{48}+4\cdot\frac{3}{72}+\frac{4}{32}+\frac{4}{48}+6\cdot\frac{5}{96}+\frac{7}{72}$
19	0.71	0	96	$2\cdot\frac{1}{32}+5\cdot\frac{1}{48}+2\cdot\frac{2}{48}+2\cdot\frac{3}{32}+\frac{5}{32}+5\cdot\frac{5}{96}+2\cdot\frac{7}{96}$
20	0.47	0	288	$2\cdot\frac{1}{18}+3\cdot\frac{1}{48}+\frac{2}{72}+\frac{3}{32}+2\cdot\frac{3}{48}+2\cdot\frac{3}{72}+\frac{3}{96}+\frac{4}{72}+6\cdot\frac{5}{96}+\frac{7}{72}$
21	0.72	0	288	$\frac{1}{18}+5\cdot\frac{1}{48}+\frac{3}{32}+\frac{3}{48}+4\cdot\frac{3}{72}+\frac{4}{72}+7\cdot\frac{5}{96}+\frac{7}{72}$
22	0.52	0	288	$8\cdot\frac{1}{48}+\frac{2}{96}+6\cdot\frac{3}{72}+6\cdot\frac{5}{96}+\frac{9}{36}$
23	0.47	0	288	$4\cdot\frac{1}{48}+4\cdot\frac{1}{72}+\frac{2}{48}+2\cdot\frac{3}{48}+2\cdot\frac{3}{72}+\frac{3}{96}+\frac{5}{72}+7\cdot\frac{5}{96}+\frac{7}{48}$
24	1.07	1	288	$2\cdot\frac{1}{48}+6\cdot\frac{1}{72}+6\cdot\frac{3}{48}+2\cdot\frac{3}{72}+8\cdot\frac{5}{96}$
25	0.53	0	288	$\frac{1}{12}+7\cdot\frac{1}{48}+8\cdot\frac{3}{72}+\frac{3}{96}+\frac{4}{96}+7\cdot\frac{5}{96}$
26	0.76	0	288	$8\cdot\frac{1}{48}+\frac{1}{96}+9\cdot\frac{3}{72}+7\cdot\frac{5}{96}+\frac{8}{96}$
27	0.54	0	380	$\frac{1}{19}+7\cdot\frac{1}{38}+2\cdot\frac{2}{38}+3\cdot\frac{2}{76}+4\cdot\frac{2}{95}+6\cdot\frac{4}{76}+3\cdot\frac{4}{95}+\frac{5}{95}$
28	0.37	0	288	$8\cdot\frac{1}{48}+2\cdot\frac{1}{96}+\frac{2}{48}+9\cdot\frac{3}{72}+\frac{3}{96}+7\cdot\frac{5}{96}$
29	0.53	0	380	$\frac{1}{19}+7\cdot\frac{1}{38}+\frac{1}{76}+2\cdot\frac{2}{38}+8\cdot\frac{2}{76}+\frac{3}{95}+8\cdot\frac{4}{95}+\frac{5}{76}$
30	0.5	0	380	$10\cdot\frac{1}{38}+10\cdot\frac{2}{95}+10\cdot\frac{4}{76}$
31	0.41	0	380	$11\cdot\frac{1}{38}+9\cdot\frac{2}{76}+\frac{4}{76}+10\cdot\frac{4}{95}$
32	0.34	0	380	$11\cdot\frac{1}{38}+11\cdot\frac{2}{76}+10\cdot\frac{4}{95}$

Downloaded from informs.org by [216.73.216.221] on 10 November 2026, at 15:47 . For personal use only, all rights reserved.

Table 1. (Continued)

n	T	Cuts	LCM	Fractions
33	1.26	0	1,824	$5\cdot\frac{1}{48}+6\cdot\frac{1}{57}+\frac{2}{38}+5\cdot\frac{2}{48}+5\cdot\frac{2}{57}+10\cdot\frac{3}{96}+\frac{4}{96}$
34	0.45	0	380	$10\cdot\frac{1}{38}+2\cdot\frac{1}{76}+9\cdot\frac{2}{76}+3\cdot\frac{2}{95}+2\cdot\frac{3}{95}+\frac{4}{76}+7\cdot\frac{4}{95}$
35	1.47	0	2,400	$10\cdot\frac{1}{48}+2\cdot\frac{2}{48}+10\cdot\frac{2}{75}+12\cdot\frac{3}{96}+\frac{5}{75}$
36	0.77	0	2,400	$7\cdot\frac{1}{48}+4\cdot\frac{1}{75}+\frac{1}{96}+5\cdot\frac{2}{48}+3\cdot\frac{2}{75}+4\cdot\frac{2}{96}+5\cdot\frac{3}{75}+7\cdot\frac{3}{96}$
37	0.83	0	2,400	$11\cdot\frac{1}{48}+\frac{1}{75}+\frac{1}{96}+12\cdot\frac{2}{75}+11\cdot\frac{3}{96}+\frac{4}{48}$
38	0.94	0	2,400	$12\cdot\frac{1}{48}+\frac{1}{75}+\frac{2}{48}+12\cdot\frac{2}{75}+12\cdot\frac{3}{96}$
39	2.42	10	7,980	$12\cdot\frac{1}{76}+\frac{1}{95}+\frac{2}{42}+\frac{2}{76}+3\cdot\frac{2}{84}+7\cdot\frac{2}{95}+8\cdot\frac{3}{84}+5\cdot\frac{3}{95}+\frac{8}{84}$
40	0.47	0	5,544	$13\cdot\frac{1}{56}+\frac{1}{77}+14\cdot\frac{2}{84}+\frac{3}{56}+4\cdot\frac{3}{77}+7\cdot\frac{3}{99}$
41	1.28	3	7,980	$14\cdot\frac{1}{76}+\frac{2}{42}+3\cdot\frac{2}{84}+9\cdot\frac{2}{95}+9\cdot\frac{3}{84}+4\cdot\frac{3}{95}+\frac{5}{84}$
42	0.32	0	7,980	$\frac{1}{38}+5\cdot\frac{1}{76}+7\cdot\frac{1}{84}+\frac{1}{95}+9\cdot\frac{2}{76}+5\cdot\frac{2}{84}+13\cdot\frac{3}{95}+\frac{4}{84}$
43	0.47	0	7,980	$5\cdot\frac{1}{76}+9\cdot\frac{1}{84}+8\cdot\frac{2}{76}+6\cdot\frac{2}{84}+15\cdot\frac{3}{95}$
44	2.02	4	319,200	$10\cdot\frac{1}{75}+2\cdot\frac{1}{76}+3\cdot\frac{1}{84}+11\cdot\frac{2}{84}+3\cdot\frac{2}{95}+\frac{2}{96}+2\cdot\frac{3}{76}+\frac{3}{84}+\frac{3}{95}+10\cdot\frac{3}{96}$

We (the class and instructors) were curious as to whether a slightly restricted form of the problem also has solutions. We modified the problem slightly so that the right-hand side of constraint (6) was set to $\lfloor n/3 \rfloor$. The model still found solutions for all n and ran slightly faster, taking around 1 second for $n = 44$.

Finally, we considered another modification to the problem where the fractions sum to $\lceil n/3 \rceil$. This problem is found to be infeasible for $n = 4$ (no candidate solutions are found) and solutions are found for all other $3 \le n \le 99$. All solutions had LCM under 1,000 and all problems took a fraction of a second to solve.

5. Conclusions

To date, all proposed approaches to solve the n-fractions puzzle have used constraint programming. Advances in solution performance have come from two fronts: the increased speed of solvers and hardware and reformulations which focus on that part of the search space with a low LCM. Here we propose an Integer Programming approach to the puzzle which also relies on the speed of modern IP solvers and hardware and focusing the search on low LCMs. Moreover, we introduce three novelties to the puzzle: integer variables to represent how many times a candidate fraction is used; lazy constraints to ensure the fractions add to 1 using exact arithmetic; and a linearisation, using logarithms, to calculate the LCM. It is likely that this model would have completed the puzzle even on the solvers and hardware available in 2009.

As a teaching case this puzzle exposes intermediate to advanced students to several useful concepts. These include: using variables that represent an aggregate quantity (the whole fraction in this case) in order to reduce symmetry; being wary of floating point rounding errors; adding lazy constraints to cut off solutions which are unacceptable for some reason; taking the log of both sides to reduce products to linear constraints; and using possibly superfluous constraints to restrict the solver to interesting parts of the search space. Our teaching experience with the case means we now use it every year.

References

Codish M (2018) A SAT encoding for the n-fractions problem. Preprint, submitted July 2, https://arxiv.org/abs/1807.00507.

Frisch A, Jefferson C, Miguel I, Walsh T (2018) CSPLib problem 041: The n-fractions puzzle. Accessed June 8, 2026, http://www.csplib.org/Problems/prob041.

Google (2024) Google OR-Tools reference. developers.google.com/optimization.

Gurobi (2024) Gurobi optimizer reference manual. http://www.gurobi.com/documentation/current/refman/index.html.

Jefferson C, Miguel I, Hnich B, Walsh T, Gent IP (1999) CSPLib: A problem library for constraints. http://www.csplib.org.

Malapert A, Provillard J (2018) Puzzle—Solving the n-fractions puzzle as a constraint programming problem. *INFORMS Trans. Ed.* 19(1):1–55.

Python Software Foundation (2024) Python fractions. docs.python.org/3/library/fractions.html.

Schulte C, Smolka G (2004) Finite domain constraint programming in Oz: A tutorial. http://mozart2.org/mozart-v1/doc-1.4.0/fdt/node31.html#section.propagators.fractions.