4OR (2022) 20:333–345
https://doi.org/10.1007/s10288-021-00488-9

EDUCATIONAL PAPER

Teaching OR: automatic evaluation for linear programming modelling

Hadrien Cambazard[1] · Nicolas Catusse[1] · Nadia Brauner[1] · Pierre Lemaire[1]

Received: 21 November 2020 / Revised: 23 June 2021 / Accepted: 25 June 2021 /
Published online: 13 July 2021

Abstract

Learning how to model a problem described in natural language as a linear program requires students to practice using various and numerous exercises. Moreover, immediate feedback on the validity of their solutions helps a better and faster understanding. In this paper, we present an idea on how students and teachers can automatically evaluate linear programming models. We also describe how this idea was implemented on the learning platform caseine.org and is now used by hundreds of students from various universities.

Keywords Linear Programming · Modelling · Training · Students

1 Introduction

Learning how to model a problem described in natural language as a linear program (LP) requires students to practice using various and numerous exercises. Moreover, immediate feedback on the validity of their solutions helps a better and faster understanding. In this paper, we present an idea on how students and teachers can automatically evaluate linear programming models. We also describe how this idea was implemented on the learning platform caseine.org[1] and is now used by hundreds of students from various universities.

[1] See a 3-minute tour of the platform on the main page of caseine.org.

✉ Nadia Brauner
Nadia.Brauner@grenoble-inp.fr

Hadrien Cambazard
Hadrien.Cambazard@grenoble-inp.fr

Nicolas Catusse
Nicolas.Catusse@grenoble-inp.fr

Pierre Lemaire
Pierre.Lemaire@grenoble-inp.fr

[1] CNRS, Grenoble INP, G-SCOP, University of Grenoble Alpes, 38000 Grenoble, France

Section 2 presents the platform Caseine which hosts our new learning tool. Section 3 states the issue we are considering, and describes the proposed solution. Section 4 shows the student view of a modelling exercise with a detailed example. Then, Sect. 5 illustrates how we generate tests used for automatic evaluation. Section 6 describes how this modelling system was implemented and used with examples of exercises and statistics on their usage. Section 7 opens to other tools for LP training, and to further features for modelling evaluation. The paper concludes on how to join the project.

2 The background: a platform for OR education

Caseine is a learning platform offering advanced learning mechanisms and content for Operations Research (OR) education. On Caseine, a community of OR teachers shares ideas, advanced tools for evaluation and a wide variety of content. The goal is to stimulate students' learning and autonomy while improving the quality of the time the teacher dedicates to them.

Caseine is based on Moodle,[2] a widely used Learning Management System (LMS), which provides students with a learning environment and let teachers monitor students' progress. It is aimed simultaneously at teachers who create courses and content and monitor students' progress, at institutions which can manage groups, and at students for their training. Compared to a classic Moodle instance, Caseine is original in that it offers a single environment where, on the one hand, students can benefit from an automatic evaluation of their mathematical models and programming codes[3] and on the other hand, teachers can create self-assessed activities, share them with the community and use shared activities.

While other learning platforms exist, we do not know of any open platform offering at the same time (1) a pedagogical environment for university training where each teacher can build their own course and (2) coding exercises with automatic and self-evaluation and (3) all sorts of pedagogical activities shared by an open community of teachers. Other platforms only address one or two of these three aspects.

There are platforms to share pedagogical activities. Wims[4] is an open platform with collections of self-assessed exercises that can be organized by teachers into courses and proposed to students. The Unisciel project, and especially its showcase platform Socle,[5] offers open courses based on Moodle with exercises that can be used by teachers to organize their own courses in their own Moodle platform. The Class'Code project[6] is composed of a community of teachers who shares activities and ideas for computer science teaching for kids up to high school. The IUTenligne project[7] offers a resource catalogue that one can export and insert into their own LMS. However,

[2] The Moodle project website https://moodle.org.

[3] Using the VPL plugin (The Virtual Programming Lab plugin for Moodle https://moodle.org/plugins/mod_vpl.)

[4] Wims: https://wims.unice.fr.

[5] Socles 3: https://socles3.unisciel.fr.

[6] Pixees: https://pixees.fr/classcode-v2.

[7] IUT en ligne: http://www.iutenligne.net.

as far as we know, these platforms do not offer exercises for code evaluation in any language.

Many other platforms offer coding exercises with automatic evaluation. For instance, on the very famous code.org[8] project based on Scratch language, teachers can create exercises or use existing ones and learners can train on exercises. This is probably the closest platform to Caseine but it mostly targets school-aged learners and it is not dedicated to university training. We can also mention FranceIOI[9] which offers programming and algorithmic training with self-assessed exercises also for school-aged learners. Other platforms propose online coding challenges to improve skills in coding and support the hiring by companies like CodinGame, Codility, Codecademy, Hackerrank.[10] However, they do not offer free sharing of exercises for university courses.

The previously mentioned platforms essentially provide resources for programming or for fundamentals of mathematics, e.g. arithmetic, calculus and algebra. Some websites offer online solvers to solve one's own modelling examples, either in connection with a book[11] or as a showcase of a commercial editor e.g. AMPL.[12] To the best of our knowledge, no platform provides rich contents on mathematical optimization in general and automatic evaluation for operations research training in particular, as caseine does.

Caseine currently offers courses on general OR tools and topics like Linear Programming (modelling, Simplex algorithm), Duality (practical interpretation, weak and strong duality theorems, sensitivity analysis), Mixed Integer Programming (modelling, Branch and Bound), Dynamic Programming, Graph Theory, Constraint Programming, etc. It also contains specialized courses like production planning and logistics and practical case studies. Apart from OR, Caseine is used in various courses in Computer Science, Industrial Engineering and Mathematics from secondary school to master programs. This paper focuses on original ideas for teaching Linear Programming modelling.

Teachers from various universities in the world use the platform with their students. Indeed, Caseine is an open academic tool, free for non-commercial use, and any teacher or student can straightforwardly connect to the platform with their own academic login, as long as their university is a member of the worldwide Edugain network.[13] Otherwise, students or teachers can request a manual account.

During academic year 2019-2020, the platform was used by 7000 users: up to 1200 users per day were connected during the COVID-19 lockdown period in France, from March to May 2020. The LP exercises are shared in 6 to 19 different courses (depending on the difficulty and interest of the exercise) for hundreds of students (up to 840 students for the most classic exercises). All teachers have their own private course

[8] Code.org: https://code.org.

[9] France-IOI: http://www.france-ioi.org.

[10] CodinGame: https://www.codingame.com, Codility: https://www.codility.com, Codecademy: https://www.codecademy.com, Hackerrank: https://www.hackerrank.com

[11] Online-optimizer: https://online-optimizer.appspot.com

[12] AMPL: https://ampl.com/.

[13] Edugain network: https://edugain.org.

for their students with all right reserved content. Teachers can access the shared space to add shared exercises to their own course. They can also choose to share content that they create with the teaching community through the sharing space, under a Creative Commons license of their choice.

3 How to evaluate an LP model?

Let's first present how students can train on modelling a problem described in natural language as a linear program. Consider the following very simple jam production problem:

We want to produce rhubarb and strawberry jam: A pot of rhubarb jam requires 1 kg of rhubarb and 3 kg of sugar and the profit is €3 per pot. A pot of strawberry jam requires 2 kg of strawberries and 2 kg of sugar and the profit is €4 per pot. The available quantities are 4 kg of rhubarb, 12 kg of strawberries and 18 kg of sugar. We want to maximize the profit.

A possible reference linear program is as follows:

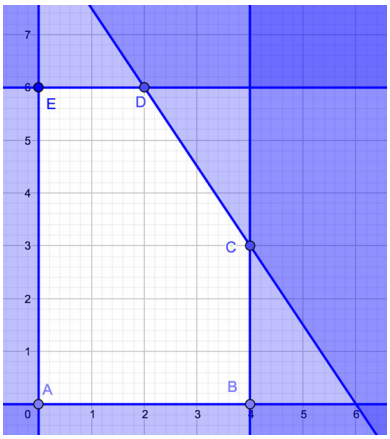
$$\begin{array}{lrcl} \max z = & 3x_r + 4x_s & & \\ s.t. & x_r & \leq & 4 \\ & 2x_s & \leq & 12 \\ & 3x_r + 2x_s & \leq & 18 \\ & x_r, \quad x_s & \geq & 0 \end{array}$$

where x_r and x_s represent the number of pots of rhubarb jam and of strawberry jam to be produced respectively.

From the description of the problem in natural language, students propose their own models. The formulations given by the students might differ from the formulation of the reference model and still be perfectly correct. We do not want to impose modelling decisions like the variables, their name, the order of the constraints, the number of variables and/or constraints, etc. Thus, numerous models are usually possible for the very same problem resulting in different polyhedral feasible regions since the space (the variables) might be different. See e.g. the model proposed by the student in Fig. 2 which is valid for the jam problem (except the right hand side equal to 17 which should be a 18) but which is neither defined on the same variables nor the same constraints.

Our objective is to assert the validity of the student's model and provide useful feedback to the student. A model is considered valid if it verifies the following conditions: it is a linear program (linear objective function and constraints); there are binary or integer variables only if allowed; the objective function is in the correct direction (denoting a maximization problem, for example); the optimal value of the objective function is the same as that of the reference model and the polyhedron associated to the constraints correctly describes the set of feasible solutions.

The first three items are classic and easy to verify with the model of the students: by asking the model, the system verifies that all variables are continuous, that the objective function and the constraints are linear and by parsing the model it verifies



a

Optimal point	c_1	c_2	x_r	x_s	Optimal value $\overline{z} = \sum c_i x_i$
A	-1	-1	0	0	0
B	1	-1	4	0	4
C	4	2	4	3	22
D	3	4	2	6	30
E	-1	2	0	6	12

b

Fig. 1 **a** The reference polyhedron for the jam problem and **b** possible values for c to verify its extreme points

the existence of non-linear reifications operators: OR, AND, IMPLIES, ABS, IF; and that the objective function is in the expected direction (min or max).

Verifying the polyhedron is challenging. We propose to evaluate the student's model by parameterizing some data of the problem. For given parameters we check that the optimal values of the reference model and of the student's model are the same. In fact, we parameterize the objective function coefficients as follows: For each extreme point of the reference polyhedron, we choose coefficients of the objective function for which this point is optimal. We also choose coefficients so that the objective function is parallel to the constraints and we test some random coefficients. The set of coefficients can be generated once, offline for all extreme points of the reference formulation. Then, it can be restricted since the number of such extreme points may be exponential in the number of variables and thus require too much time or energy when the students launch the test.

We detail how to test the student model in each vertex of the reference polyhedron. The only technical requirement placed on the student is to use a vector c that contains some parameters of the problem (in our case, the coefficients of the objective function). For instance, in the jam problem, the student has to use $c = (3, 4)$ in the model which means that the cost of a pot of rhubarb jam has to be written c_1 instead of 3 and the cost of a pot of strawberry jam has to be written c_2 instead of 4. Note that these coefficients are always defined by the input data and not the model.

For the reference model, we adapt similarly the objective function to $c_1 x_r + c_2 x_s$. The polyhedron associated to the constraints of this very simple model is given in Fig. 1a. To verify the constraints of the student, the system will make the tests with the values of c given in Fig. 1b. For instance, when $c_1 = 3$ and $c_2 = 4$ the optimal point is D with value 30 and the system verifies that the student's model also returns 30 for these values of c_1 and c_2. The values of c_1 and c_2 are arbitrary but allow to test all five extreme points of the polyhedron. Their calculation is described in Sect. 5. Vector c is replaced in the model of the student while parsing the file.

Ideally, we would also like to do the opposite, *i.e.* check the objective value given by the reference model at each vertex of the student's polyhedron. However, we have no control over the model produced by the students and the generation of such extreme points could be time-consuming. We therefore opted for a pragmatic alternative: to evaluate also the student's code with objective directions that are orthogonal to the constraints of the reference model as well as a number of random directions (this number is specified by the teacher to ensure enough robustness for detecting incorrect models).

Note that the correctness of the optimal solutions themselves is not further verified, e.g. by a solution checker: this would require to specify a solution description format and would add extra burden both on the teachers and students to conform to it. In our experience, this is only a plus for very large problems and models and the proposed approach turns out to be already very efficient in practice.

Besides, for real classes, we manually validate the mathematical description of the model since it could be valid but nevertheless abstruse, clumsy, too complicated, etc. and this is not verified automatically. The automatic tests allow to gain time for the teacher and to have immediate feedback for the student but they do not exempt the teacher from some proofreading of the code quality. However, the time dedicated by the teacher to the student is of better quality.

4 The student view of a modelling exercise

To create a modelling exercise, the teacher provides the description of the problem in natural language and the reference model. The exercise is then generated automatically. We use an external program to generate the tests described above before we add them into the evaluation system. Teachers can also manually add other tests, and they can modify the generated tests.

The student view is given in Fig. 2. In this example, the students have to model the jam problem with the modelling language OPL of IBM ILOG.[14] They can run Cplex Community Edition solver on the model by clicking the rocket button. They can evaluate their solutions by clicking the check mark button. The editor contains the student's program. The right part of the window displays the evaluation with the 5 tests described in Fig. 1b, the 3 tests with an objective function parallel to the constraints (with hidden parameters so that the student cannot guess the constraints from the tests) and a randomized test. For the extreme point tests, namely, the first 5 tests, the students can see the values of c used and the outputs of their model and of the reference model. The linearity of the model, the type of the variables and the direction of the objective function are also verified. The proposed grade depends on the number of tests with equal objective values for both models. Seeing these results, the student can decide whether their model is under-constrained or over-constrained.

The students can work in the web interface but they can also use an external editor with the Caseine plugin, see e.g. Fig. 3 for the OPL Studio IDE. The plugin allows to

[14] IBM ILOG CPLEX Optimization Studio 20.1.0 documentation https://www.ibm.com/support/knowledgecenter/SSSA5P_20.1.0/COS_KC_home.html.

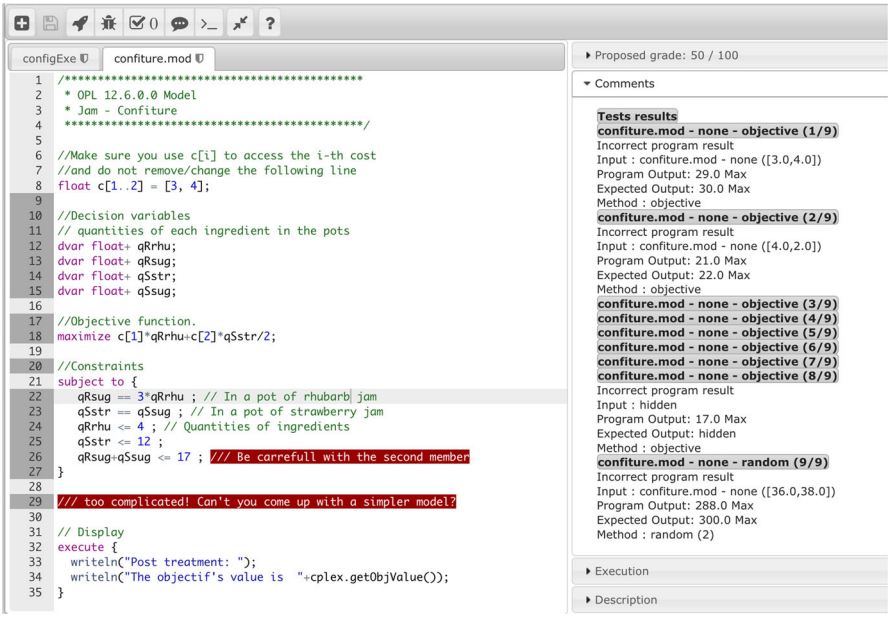


Fig. 2 Student view of a modelling exercise

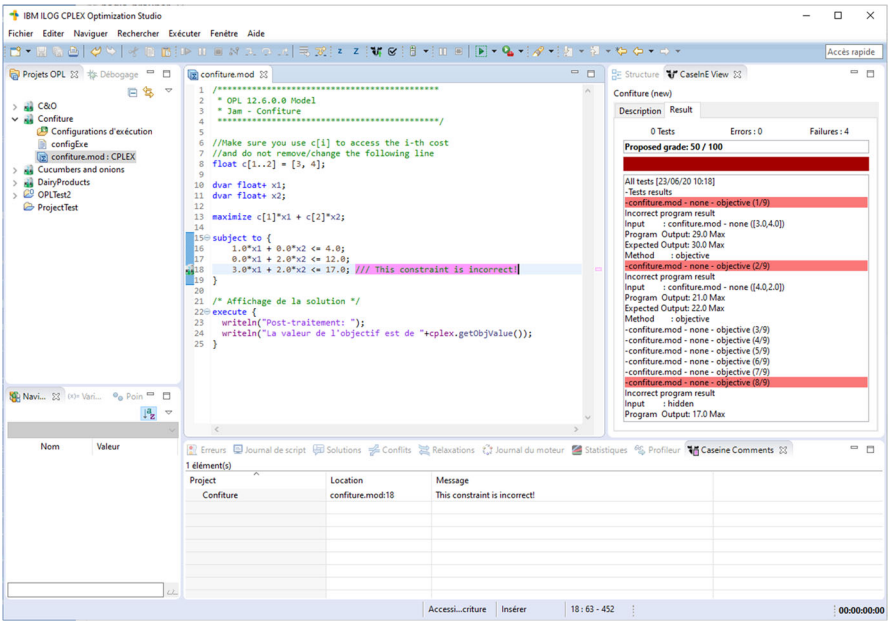


Fig. 3 OPL Studio IDE with the Caseine view

pull the description of the exercise and to push the student's solution into the platform. It also allows to launch the evaluation and it proposes a view for the evaluation result.

5 Automatic test generation: on the teacher's side

The automatic generation of tests is based on the reference model. The model contains the variables x_i for $i = 1, 2 \ldots n$, the objective function which is of the form max / min $\sum_{i=1}^{n} c_i x_i$ and some inequalities and equalities and sign constraints on the variables. We now describe how the tests are generated.

First, add positive slack variables thus transforming each inequality into an equality. If the j-th inequality is of the form $\sum_{i=1}^{n} a_{ij} x_i \leq b_j$, it becomes $\sum_{i=1}^{n} a_{ij} x_i + s_j = b_j$; otherwise (namely $\geq$ inequality) it is transformed into $-\sum_{i=0}^{n} a_{ij} x_i + s_j = -b_j$. One then obtains a system of the form $Ax = b$ with the initial equalities and the transformed inequalities. Then we enumerate all possible bases of this system and verify that the associated basic solution is feasible (with the original sign constraints and positivity of the added slack variables).

Next, for each feasible basic solution, we calculate a vector of coefficients c for the variables such that this basic solution is the only optimal solution. For this purpose, we know that a basic solution is optimal for a given objective function if the reduced costs of the basic variables are zero and the reduced costs of the non-basic variables are all negative (for a maximization problem). Thus, given a basic solution with indices in B for the basic variables and indices in N for non-basic variables, the following objective function is maximal at this basic solution:

$$z = -\sum_{i \in N} x_i - \sum_{j \in N} s_j + \overline{z} = \sum_{i=1}^{n} c_i x_i$$

Note that the coefficients -1 are arbitrary and any negative values would be suitable as well; $\overline{z}$ is a constant equal to the objective value reached at this vertex with the adequate c_i coefficients; it is given in the last column of Fig. 1b.

To find the coefficient of x_i (*i.e.* find c_i), one has to eliminate the slack variables s_j by replacing them by $b_j - \sum_{i=1}^{n} a_{ij} x_i$ or $-b_j + \sum_{i=1}^{n} a_{ij} x_i$ depending on the direction of the original inequality. For simplicity, we only consider the first case for s_j. Hence the objective function is of the form:

$$z = \sum_{i \in N} \left(-1 + \sum_{j \in N} a_{ij} \right) x_i + \sum_{i \in B} \sum_{j \in N} a_{ij} x_i + \left(\overline{z} - \sum_{j \in N} b_j \right)$$

and thus

$$\begin{cases} c_i = -1 + \sum_{j \in N} a_{i,j} & \text{if } x_i \text{ is non basic, } i.e.\ i \in N \\ c_i = \sum_{j \in N} a_{i,j} & \text{if } x_i \text{ is basic, } i.e.\ i \in B \end{cases}$$

Applied to the jam example, with $\overline{z} = \sum_{j \in N} b_j$, this leads to the coefficients described in Fig. 1b.

The directions of the constraints are also tested but with hidden parameters so as not to give too much information to the students.

Notice that it is not always possible to test all extreme points with this method. This is the case if, due to specific features of the problem description, the objective function of the reference model does not contain all the variables. Consider for instance the reference model $\{\max c_1 x$ such that $x + y \leq 1; x \geq 0; y \geq 0\}$: the polyhedron has 3 extremal points but by varying the value of c_1, one can not distinguish between points (0,1) and (0,0). The same problem appears if two variables share the same parameterized coefficient in the objective function. Consider for instance the reference model $\{\max c_1(x + y)$ such that $x \leq 1; y \leq 1; x \geq 0; y \geq 0\}$: the polyhedron has 4 extreme points but only 2 can be tested. One has to be aware of these limitations when evaluating students' models.

More generally, the users have to keep in mind that if the system indicates a mistake, then the model is incorrect but if the system can find no mistake, it does not necessarily mean that the model is valid.

6 A collection of shared exercises

Once created, an LP modelling exercise can be shared with the community. Indeed, the platform contains a collection of exercises presented by increasing difficulty. For each exercise, the level estimated by the teacher is indicated by a color from green or blue for easy exercises to red or black for harder ones. All teachers can contribute to the collection by sharing their own exercises, and use any shared exercise in their own courses. Students can give a feedback of their interest for each exercise through the reaction system, see Fig. 4.

Table 1 describes the use of some of these exercises. Line *number of courses* shows that the exercises are used in several courses from various universities. Line *number of users* shows the number of users who tried the exercise and *number of submissions* shows the total number of submissions for each exercise. The next line, *average number of submissions* is the quotient of the two previous lines and is an indicator of the difficulty of the activity. Line *number of users with 100%* indicates the number of users who finished the exercise successfully. The last line, *first submission* indicates the age of the exercise. The linear programming exercises can have various types of modelling difficulties:

- very simple models like the previous jam production example (e.g. Cucumber and Onions);
- a slack variable is not explicit in the description and is used in the objective function (e.g. Dairy products);
- flow models;
- finding the right variables is already an issue (e.g. Apples);
- the heterogeneity of the units of the variables;

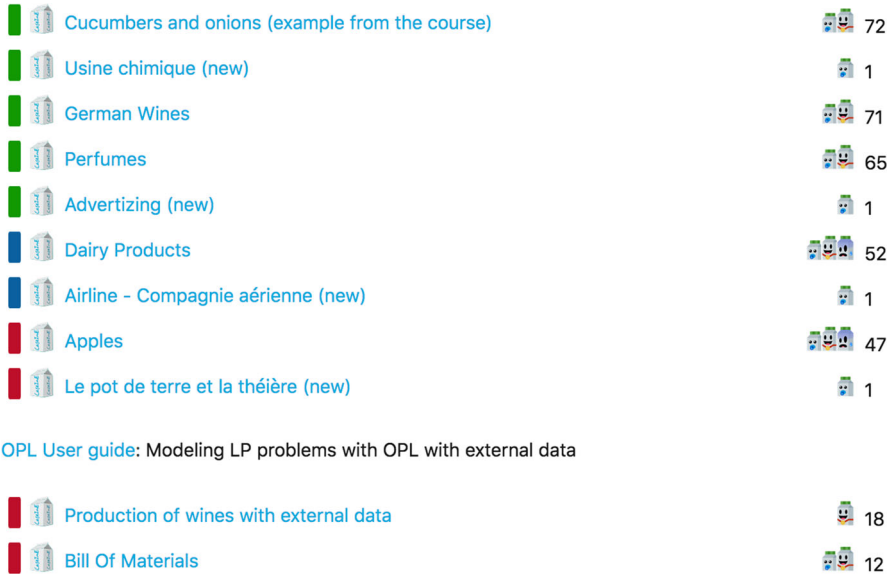


Fig. 4 Modelling exercises with difficulty level (color rectangles) on the left and student's reactions on the right

Table 1 Usage of popular linear programming modelling exercises (July 2020)

	Cucumber and onions	Dairy products	Apples	wine production
Number of courses	19	16	8	6
Number of users	840	795	470	297
Number of submissions	4447	8793	6617	5312
Average nb of submissions	5.2	11.1	14.1	17.9
Number of users with 100%	756	649	356	232
First submission	05/2016	07/2019	08/2016	09/2018

- complexity comes from the length of the description (industrial case studies with many/various constraints and constraint types);
- the difficulty also increases in the model complexity from the in-extenso description to an abstract model with external data and the use of keywords like *sum* or *for* (e.g. Wine Production);

Some exercises concern mixed integer programming from classic combinatorial problems (Knapsack, Bin Packing. . .) and classic graph problems (Independent Set. . .) to real based case studies (a Power Plant Design).

7 Conclusion and perspectives

To conclude, we first discuss the integration of such automated exercises within a complete course on linear programming: in Sect. 7.1, we present other more or less classic resources for LP training. Then, in Sect. 7.2, we describe current and future developments on our modelling evaluation tool. We then end the paper, Sect. 7.3, with a discussion on how to participate in the project.

7.1 Other resources for LP training

The Caseine platform offers other resources for LP training: the *One idea, one story* pages, which put the results into a historical perspective, or a shared database of numerous and various questions going from the very basics to advanced notions. Those teaching tools focus on the student active role by increasing their engagement and autonomy. On the platform there is a large variety of usages of these tools from complete autonomy (personal/team work, at home/during the course) to traditional classroom with validation in autonomy or as a support for reverse teaching.

Creating technologically advanced questions is time consuming. Therefore the idea is to share them within a community. This also improves the quality and visibility of the exercises since sharing content implies a benevolent peer-reviewing. Technically, these questions are merely managed as classic Moodle questions. However, we emphasize two original uses of this bank. Firstly, we can generate offline hundreds of questions, based on dedicated patterns (see e.g. Fig. 5); randomly drawn questions from selected collections are then submitted to students, allowing them to train on always renewed tests. Secondly, the thousands of LP-related questions is generated and shared by many teachers from various universities who use them in their own courses.

The *OR open courses* (https://moodle.caseine.org/course/view.php?name=LPOpen) illustrate the usage of automatically evaluated modelling exercises along with quizzes built from the bank of questions. They show how modelling exercises and quizzes both fit well in the setting of a course, as they complement one another, allowing different assessments for different pedagogic purposes.

7.2 Perspectives: going further with tests

We are currently developing a new tool to enhance the experience of automatically evaluated modelling exercises. This tool finds which constraints are incorrect based on the errors in the student's solution. The general idea is: because each constraint is tight for a specific set of points, if the evaluation fails on all these points, it is likely that the corresponding constraint is incorrect. Thus, we can give a better feedback to the students. For instance, in the jam problem, if evaluations at all points are correct except for C and D, then the system indicates that the mistake is probably in the sugar constraint.

In this paper, we declare parameters for a subset of the problem data, namely the objective function. In other contexts, it is usual to test various instances of a problem, and hence to change the whole data: in an industrial context, one may test

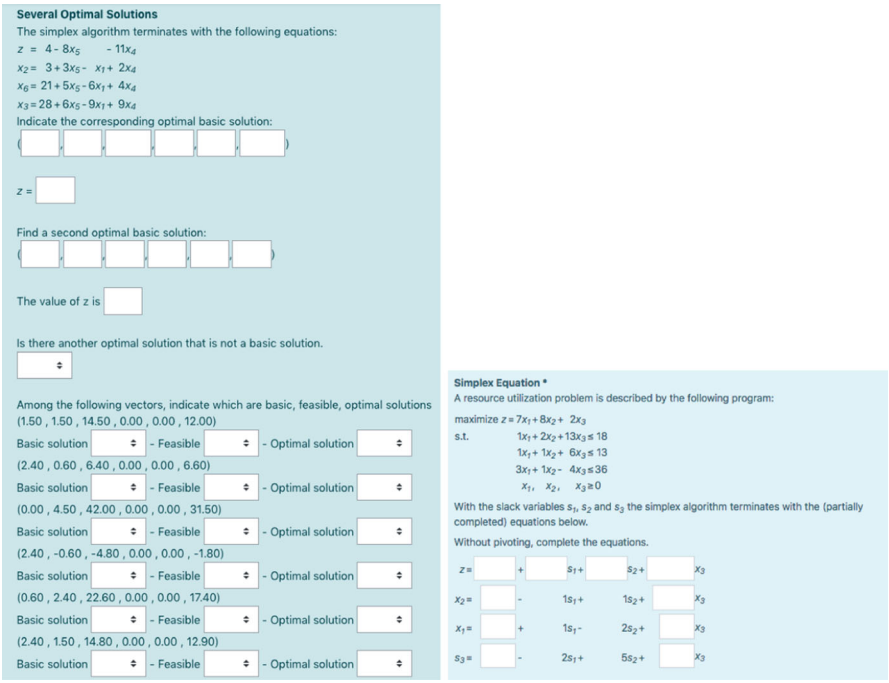


Fig. 5 Two examples each drawn from a collection of 200 questions

with several historical data sets to verify the validity or response time of a model, and in a research context, one may test various random or benchmark data sets to assert the efficiency of a formulation. This could be done but our goal is different: we want to give students the best possible feedback to improve their training. By limiting the changes to the objective function, we ensure that we can provide a clear and understandable feedback. Changing other coefficients would be technically easy, but requires further investigations to put a helpful diagnosis on the mistakes. Even though the procedure described in this paper for testing linear program works very well in practice, it is not sufficient for mixed integer programs (MIP), for which clever instances have to be designed to test correctly the students' model.

Regarding MIP, we are also working on additional automatic evaluation features. For small models, we verify the number of feasible solutions using a constraint programming model. It could be interesting to try and work with the convex hull of the feasible solutions.

Among other future works, there is a need for an automatic test selection, when the system generates too many tests for large models (this is currently done by hand), and for integrating other modelling languages (only OPL models are currently supported).

7.3 How to join

In this paper, we presented an original idea on how to automatically evaluate the linear programming modelling skill of a student. We explained how we implemented it in the

Caseine platform and how teachers from various universities used it in their courses for hundreds of students. The objective of this tool is to provide students with a better learning and to increase the efficiency of teachers.

In the context of the Covid-19 pandemic, the community of Caseine teachers was happy to rely on this tool (among others) for distant learning even if it was not designed especially for it. The teacher community was very active for sharing content, experiences and ideas. We observed that teachers who were already users of the Caseine platform relied even more heavily on the platform during the lockdown period. During that time, Caseine saw a surge of activity: the activity level in the first month of lockdown (from March 15 to April 15, 2020) is similar to the cumulated activity of the 5 previous months.

If you might be interested in using these tools or joining the adventure, you can start by visiting the site caseine.org.

Acknowledgements The authors would like to thank the two engineers of the platform, Astor Bizard and Florence Thiard who develop the tools and help teachers use the platform. They also thank Bernard Penz and Olivier Briant for interesting discussions. They thank Michael Perin for the discussions which lead to the first idea described in the perspectives.

Compliance with ethical standards

Funding This work has been partially supported by the Idex Université Grenoble-Alpes (ANR-15-IDEX-0002) and by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01), both funded by the French program Investissement d'avenir.

Conflicts of interest The authors declare that they have no conflict of interest.

Availability of data and material All functionalities described in this paper are available on the Caseine platform: caseine.org

Code availability All examples described in this paper can be tested on the Caseine platform: caseine.org