Yugoslav Journal of Operations Research
31 (2021), Number 1, 23–44
DOI: https://doi.org/10.2298/YJOR191115019M

THE RaPID-Ω SYSTEM: ROOM AND PROCTOR INTELLIGENT DECIDER FOR LARGE SCALE TESTS PROGRAMMING

Fernando A. MORALES
Escuela de Matemáticas. Universidad Nacional de Colombia, Sede Medelín.
Carrera 65 #59A–110, Medellín, Colombia
famoralesj@unal.edu.co

Received: November 2019 / Accepted: June 2020

Abstract: We present the documentation and mathematical modeling of the open-box system RaPID⁄Ω. The software is designed for the choice of classrooms and the assignment of proctoring duties in massive tests, which is a common situation in educational institutions offering large coordinated lower division courses. The mathematical model is a binary integer programming problem: a combination of the 0-1 knapsack problem and the job-assignment problem. The system makes decisions according to the following criteria in order of priority: minimization of labor-hours, maximization of equity in the distribution of duties and maximization of the proctoring quality.

Keywords: Open-box Software, Python Program Documentation, Binary Integer Programming.

MSC: 90C10, 90B80, 68N15 .

1. INTRODUCTION

In this work we present the documentation and mathematical modeling of the open-box system RaPID⁄Ω, designed to optimize the rooms choice and the proctor scheduling for the logistics of massive tests. This type of software is needed in educational institutions having large coordinated courses (specially for the lower division ones) with simultaneous common tests. The proposed software was constructed based on the needs of a specific case, *Escuela de Matemáticas* (School of Mathematics) at *Universidad Nacional de Colombia, Sede Medellín*

Table 1: Historical Enrollment Table

Semester	DC	IC	VC	VAG	LA	ODE	NM	Total
2010–1	1631	782	381	1089	983	668	142	5676
2010–2	1299	1150	427	1003	1007	562	261	5709
2011–1	1271	1136	512	1078	900	663	269	5829
2011–2	951	850	513	652	812	1170	289	5237
2012–1	1619	1096	559	1110	1116	752	366	6618
2012–2	1486	1190	601	1076	1144	825	356	6678
2013–1	1476	1044	604	1231	1037	902	319	6613
2013–2	1446	1212	549	1187	1103	786	326	6609
2014–1	1460	1184	676	1192	1000	890	295	6697
2014–2	1399	1126	564	1198	1012	695	234	6228
2015–1	1097	925	565	1076	793	601	201	5258
2015–2	1797	1214	605	1314	1099	808	274	7111
2016–1	1675	1323	582	1549	1017	950	263	7359
2016–2	1569	1296	594	1355	1009	1019	284	7126
2017–1	1513	1315	515	1088	798	736	134	6099
Mean	1445.9	1122.9	549.8	1146.5	988.7	801.8	267.5	6323.1

(National University of Colombia at Medellín). Hence, the simple model examples we present are based on our study case.

In contrast with black-box commercial software, this open-box tool is aimed to be easily used and/or modified to the needs of other programming scenarios (most likely other educational institutions with different but similar testing procedures). The School of Mathematics is part of the College of Science within the National University of Colombia at Medellín, it teaches two types of courses: specialization (advanced undergraduate and graduate courses in mathematics) and service courses (lower division) for the whole University. The latter are: *Differential Calculus* (DC), *Integral Calculus* (IC), *Vector Calculus* (VC), *Differential Equations* (ODE), *Vector & Analytic Geometry* (VAG), *Linear Algebra* (LA), *Numerical Methods* (NM), *Discrete Mathematics* (DM), *Applied Mathematics* (AM) and *Basic Mathematics* (BM, college algebra). The total demand of these courses amounts to an average of 7200 enrollment registrations per semester. The last three courses, DM, AM, BM, do not test their students in a coordinated fashion but independently, i.e., each lecturer designs his/her own evaluation method. Consequently, they will not be subject to this analysis. Given that most of the students attending the National University of Colombia at Medellín pursue degrees in Engineering, the courses DC, IC, VC, ODE, VAG, LA and MN are massive and pose significant logistic challenges for booking their respective evaluations; see Table 1 below. On a typical semester, these courses are divided in sections (between

8 and 22, depending on the enrollment) of sizes ranging from 80 to 140 (because of classroom seat capacities). The evaluation consists of three exams, which the students take simultaneously; the personnel in charge of proctoring duties consists of approximately 45 lecturers among tenured and adjunct faculty, as well as 70 teaching assistants among graduate and undergraduate students. Moreover, the Teaching Assistants and Adjunct Faculty are not full-time employees ergo, they introduce significant time constraints in the task assignment due to their schedule. Typically, each of the coordinated courses takes three tests during the semester, therefore three rounds of tests need to be scheduled each semester. Currently, each round selection of rooms and proctoring duties assignment are decided with the RaPID$\not{}\Omega$ system.

The RaPID$\not{}\Omega$ system approaches the problem in three steps. First, the module **Room_Decision.py** minimizes the number of needed proctors in each examination activity in order to remove unnecessary labor-hours. Second, the **Personnel_Decision.py** module maximizes the equity in the proctoring hours among the personnel. Third, the **Crew_Organization.py** module optimizes the proctoring quality according to the experience record of each proctor. Furthermore, the decisions are made with that order or priority. The system is implemented in Python 3.4, it uses libraries such as pandas (Python Data Analysis Library) and SciPy. RaPID$\not{}\Omega$ runs from command line, and it can be freely downloaded from

`https://sites.google.com/a/unal.edu.co/fernando-a-morales-j/home/research/software`

The rest of the paper is organized as follows: in Section 2 the input datasets are exposed, together with their meaning and format; in Section 3 the output files are presented, two of these files are final while two are intermediate. Section 5 exposes the modeling and algorithms for the first module of the system concerned about optimal room choice and number of students in each room. Section 6 presents the modeling algorithms of the second module regarding the choice of proctors among available personnel. Section 7 presents the algorithm to decide the optimal position to proctor a test, starting from a previously scheduled crew. While any user needs to understand sections 2 and 3, sections 5, 6, and 7 are of interest only for a developer.

2. THE INPUT DATA SETS

In the present section, we describe the input data files for the RaPID$\not{}\Omega$ system, explain the contents and structure of each dataset, as well as its motivation. In our study case, a round of tests for the seven courses is typically scheduled in the span of two weeks. The input files are: Available_Rooms.xls, Room_Data.xls, Personnel_Time.xls, Proctor_Log.xls and Professors.xls. The first two are concerned with physical spaces, while the last three accounting for human resources.

2.1. The Available_Rooms.xls file

The first input data is the spreadsheet **Available_Rooms.xls**, which contains the structure presented in the example of Table 2 below. For the School of Mathematics, the first step in programming an examination is to request the necessary seats for each activity to the University's *Office of Building Management*. The information sent is summarized in the last three rows of Table 2. Conversely, the Office of Building Management replies a list of rooms available at the time and date requested and capable of holding the necessary enrollment, see Table 2 for the example. It should be noticed that there is a slack for all the cases, i.e., the total capacity of the available rooms always exceeds the number of students: DC 1354, IC 1093, VC 626, VAG 1006, LA 626, ODE 897, MN 402. Consequently, the choice of available rooms can be done so to minimize the number of necessary proctors.

Remark 1 (Format Available_Rooms.xls). *Some format guidelines* ***must be observed*** *in the Available_Rooms.xls file for the correct functioning of RaPID$\not{\Omega}$.*

(i) *The time must include the abbreviation day, a blank space, two digits for each hour and a hyphen in between:* ***dd TT-TT****, e.g.,* ***Mo 08-10*** *instead of* ***Mo 8-10****. Of course, the time slots can be modified according to the scheduling needs (e.g. Monday from 9:00 to 12:00), as long as the format* ***dd TT-TT****, is consistently preserved through the datasets (e.g. the file Personnel_Time.xls in Section 2.3).*

(ii) *The names of the courses (acronyms or not) in the columns can be modified as long as the labels are consistent with those in the file Professors.xls. Naturally, the columns of the file can be increased or decreased according to necessity.*

(iii) *The data in the column "Room" need not be sorted.*

(iv) *The capacity of each room need not be written on the courses columns, it suffices to write "1", to indicate that the room is available for the activity. The system will actually read the value of the room capacity from the file Room_Data.xls (see Section 2.2); however, it may be desirable to write the capacity in this file, in order to check on the input spreadsheet itself, if the number of available seats is greater than or equal to the number of students.*

2.2. The Room_Data.xls file

The Room_Data.xls sheet centralizes the information about all the classrooms on campus, not only those needed for the examination activities, see Table 3 for a minimal example.

The table contains the number of seats or capacity of each room, and a column of observations where some annotations can be made such as: how the room is to be opened, or if it has accommodations for students with disabilities. This file is the most stable of all, as it changes only when the nature of the rooms changes, therefore it is less vulnerable to human error than the file Available_Rooms.xls; this is why RaPID$\not{\Omega}$ reads the capacities from this file.

Table 2: Example of Available Rooms

Room	DC	IC	VC	VAG	LA	ODE	NM
03-210	30						
04-108	29						
04-109	60						
04-110				58			
04-111							
04-206	65						
05-101							79
11-102						56	
11-124	22	22					
11-125		41					
11-202						150	
11-203		39		39		43	
11-208	50	50		50			
11-209	30	30		30			
11-225		60		60		60	
14-109	51	49					
14-232	50			50			
16-223	63	63	63	63	63		
16-224	60	72	60	72	60	58	
21-303	34			34			
21-307	52		52		52		
21-314	79	79	79	70	79		
21-320	64	64	64	64	64	58	
21-328			30				
21-331	29						
24-307						170	
25-301						90	79
41-102		106					
41-103	106		106		106	106	102
43-110				54		53	
43-111						53	
46-114				56			
46-208	47	47		47			
46-209	50	50	50		50		
46-210	52		52		52		
46-211	52	52		52			
46-212	49			49			
46-301	44	44		44			
46-303		43		43			
46-304	41	41		41			
46-307	100	100	100		100		
46-311		41					
Students	1300	1050	608	951	600	822	150
Date	30-III	01-IV	06-IV	01-IV	06-IV	02-IV	05-IV
Time	Sa 12-14	Mo 08-10	Sa 14-16	Mo 10-12	Sa 12-14	Mo 10-12	Mo 08-10

2.3. The Personnel_Time.xls file

The third input data for the system is the spreadsheet **Personnel_Time.xls**, it has the structure presented in the minimal example of Table 4 below. As explained

Table 3: Example of Room Data

Room	Capacity	Observations
03-210	32	Doorkeeper
04-108	28	Key
04-109	70	Access Code
04-207	70	Wheelchair Ramp
05-101	80	Doorkeeper
11-102	56	Card

in the introduction, the team of proctors is not made of full-time employees, therefore, there are time constraints when scheduling a test. This is particularly acute in the case of the Teaching Assistants whose labor duties amount to 10 hours per week and their academic duties may be time-conflicting with the examination activities.

Table 4: Example of Time Personnel

Name	Cell	email	ID	Experience	Level	Mo 10-12	Mo 12-14	Sa 08-10
TA 1	C 1	1@m.co	ID 1	1	Undergraduate	Day Off		1
TA 2	C 2	2@m.co	ID 2	2	Undergraduate	Busy	1	1
TA 3	C 3	3@m.co	ID 3	1	Undergraduate	1	Class	1
TA 4	C 4	4@m.co	ID 4	1	Undergraduate	1	1	1
TA 5	C 5	5@m.co	ID 5	2	Undergraduate	1	NA	1
TA 6	C 6	6@m.co	ID 6	2	Postgraduate	1	0	1

The table contains fields for identification ("Name" and "ID"), contact ("Cell", "email"), and rating ("Experience" and "Level"). Finally, the availability fields are represented by multiple time slots. In this minimal example only three time-slots were included: Mo 08-10, Mo 10-12, Sa 08-10. In practice, all possible time-slots should be contained in the table, namely: Mo 08-10, Mo 10-12, ..., Mo 16-18, Td 08-10, ..., Fr 16-18, Sa 08-10, ..., Sa 16-18.

Remark 2 (Format Personnel_Time.xls). *The following instructions must be observed when building the Personnel_Time.xls sheet.*

(i) *The time slots **dd TT-TT**, must be consistent with those of the file Available_Rooms.xls (see Remark 1 (iii) Section 2.1) for the system to work properly. They can be modified according to the scheduling needs (e.g. Monday from 9:00 to 12:00), as long as the consistency between datasets and the format **dd XX-YY** are preserved.*

(ii) *The time slots columns **dd TT-TT** must indicate whether or not an individual is available. To indicate availability use "1".*

(iii) *Only **availability** indicated by the number "1" is important for later calculations. In particular, if any other information is set (e.g., "Available", "Free",* *"Busy", etc.), the system will understand that the individual is **unavailable** at that time slot.*

(iv) *There is no need to indicate unavailability. As shown in the example, the reason why a TA is not available, can be declared or not. Unavailability can also be marked with a "0" as in the example above. However, this annotations will not impact on the system.*

(v) *The column "Level" has to be filled with the words "Undergraduate" or "Postgraduate" for the system to understand the academic level of each TA. Furthermore, this information will play a key role in the greedy algorithms of the module Crew_Organization.xls, see Section 7 and Algorithm 4.*

2.4. The Proctor_Log.xls file

The **Proctor_Log.xls** is a file containing the record of proctoring duties that the TAs have served in an observation time-window, namely, a term, a semester or a year (depending on the institutional policy). Its structure is presented in the minimal example of Table 5 below and it is fairly similar to that of Personnel_Time.xls; it agrees on the columns holding each TA's information. Each of the remaining columns represents an examination activity that took place in the observation time-window, up to the programming date. For each examination event, the number "1" indicates that the individual served on it. The column "Total" indicates the total number of shifts the TA has taken so far.

Remark 3. *Some observations about this file are the following*

(i) *When the system is initiated for the first time this file has to contain all the columns holding the information of each TA ("Name", "Cell", "email", "ID", "Experience", "Level") and the column "Total" with value "0" in all its rows.*

(ii) *Once the system is executed, for the next round of tests, an updated file: **Updated_Proctor_Log.xls** will be generated automatically, see Section 3.2 and Table 9 for this file.*

Table 5: Example of Proctor Log

Name	Cell	email	ID	Experience	Level	ODE, 04-II	Total
TA 1	C 1	1@m.co	ID 1	1	Undergraduate		0
TA 2	C 2	2@m.co	ID 2	2	Undergraduate	1	1
TA 3	C 3	3@m.co	ID 3	1	Undergraduate	1	1
TA 4	C 4	4@m.co	ID 4	1	Undergraduate	1	1
TA 5	C 5	5@m.co	ID 5	2	Undergraduate	1	1
TA 6	C 6	6@m.co	ID 6	2	Postgraduate		0

2.5. The Professors.xls file

The input file **Professors.xls** is a spreadsheet with the structure of the minimal example presented in Table 6 below. Unlike the TAs, the lecturers are full-time employees, therefore they pose no time constraints when scheduled for proctoring duties. The column "Coordinator" indicates whether the Lecturer is the coordinator of the course and will be the general supervisor of the examination activity, therefore he/she will not be scheduled for proctoring testing rooms. Columns "Subject", "Subject_2" indicate which subject is to be lectured in the academic period.

Remark 4. *Some observations are*

(i) *Indicating that a faculty member is a course coordinator must be done with the word "yes". If any other character or word is set (e.g., "Yes", "1", "coordinator"), the system **will not** understand the corresponding individual as the coordinator and will include him/her in the proctoring duties as it dose with any other lecturer.*

(ii) *By default, the system will assign an instructor to proctor only the examination activities of the subject he/she is lecturing. More specifically, in Table 6 Lec 3 will be assigned to proctor only examinations of DC.*

(iii) *Some instructors may have two or more service courses assigned, however, the system takes into account only the first subject for proctoring duties, e.g., Lec 4 will be assigned to proctor only examinations of ODE and not those of AL. Therefore, the remaining subjects may be omitted.*

(iv) *If it is the User Institution's policy to make lecturers participate in the proctoring of every subject they teach, it suffices to create one more row for the second subject. For instance, Lec 6 teaches two subjects, therefore two rows should be created for he/she, one having "VAG" in the column "Subject", the other having "DC" in the same column.*

(v) *If the User Institution has the policy of having only the TAs proctoring the tests, the column "Coordinator" should be filled with the word "yes". This will suffice to exclude the instructors from the job assignment (as it does with the actual coordinator), but is important to stress that the Professors.xls file **must exist**, with the **columns** described above (even if it is empty), for RaPID$\not{}\Omega$ to work correctly.*

3. THE OUTPUT FILES

In the present section we describe the output files that RaPID$\not{}\Omega$ produces and explain its contents. From the **user's** point of view, only two files are important: **Proposed_Programming.xls**, and **Updated_Proctor_Log.xls**, which will be explained first. However, a **developer** should understand two other files produced as an intermediate step towards the final solution, from one system's module to the next, which are: **Scheduled_Rooms.xls** and **Sheduled_Crew.xls**

Table 6: Example of Professors file

Name	Coordinator	Subject	Subject_2	Cell	email
Lec 1	yes	VC		C 100	Lec1@m.co
Lec 2		NM		C 200	Lec2@m.co
Lec 3		DC		C 300	Lec3@m.co
Lec 4		ODE	AL	C 400	Lec4@m.co
Lec 5		MD		C 500	Lec5@m.co
Lec 6		VAG	DC	C 600	Lec6@m.co

3.1. The Proposed_Programming.xls file

The **Proposed_Programming.xls** file is an excel book, and it is the ultimate goal of the system. Here, there is a sheet for each course in the round. In our study case: DC, IC, VC, VAG, LA, ODE, and NM. Each sheet has the structure of Table 7. The fields are Room (code of the room), Envelope (or pack of tests), Observations, Capacity, Students, Slack, Test, Date, Proctors (number of assigned proctors), Name (name of the assigned proctors), Cell, and email.

Remark 5. *Two observations are in order*

(i) *The number of assigned proctors to a room depends on the number of students. The default value is one proctor per 54 students, which can be changed by the user at the time of executing RaPID⨍Ω, see Section 5.1, Definition 7 to change this value.*

(ii) *If two (or more) proctors are assigned to one room, two (or more) rows will be equal except for the name of the proctor (e.g., the first and second row in Table 7). As a consequence of these repetitions, the sum of the column "Students" in this file will not yield the number of students taking the test, as it happens in the example at hand.*

3.2. The New_Proctor_Log.xls file

Once the system is executed for a next round of tests, an updated file **New_Proctor_Log.xls** will be generated automatically. This will have the previous service record, and it will paste it on the left: one column per examination activity. In each column a "1" will be written if the individual was selected to serve in the corresponding activity, and also, the "Total" column will be updated.

Consider the minimal example presented in Table 9. Here, it is understood that the proctor log file is that of Table 5 (only one examination ODE on February the 4th took place before) and the system is programming only one examination in the next round, which is AVG on March the 4th. The updated log of Table 9 writes the proctoring duties for all the employees and an updated "Total" of service.

Remark 6. *The following must be observed*

Table 7: Example of Proposed Programming CV, 608 Students

Room	Envelope	Observations	Capacity	Students	Slack	Test	Date	Proctors	Name	Cell	email
41-103	1	Doorkeeper	106	106	0	VC	Sa 14-16 06-IV	2	TA 7	C 7	7@m.co
41-103	1	Doorkeeper	106	106	0	VC	Sa 14-16 06-IV	2	TA 41	C 41	41@m.co
46-307	2	Card	80	80	0	VC	Sa 14-16 06-IV	2	TA 20	C 20	20@m.co
46-307	2	Card	80	80	0	VC	Sa 14-16 06-IV	2	TA 48	C 48	48@m.co
46-209	3	Card	50	50	0	VC	Sa 14-16 06-IV	1	TA 65	C 65	65@m.co
16-224	4	Card	72	57	15	VC	Sa 14-16 06-IV	2	TA 28	C 28	28@m.co
16-224	4	Card	72	57	15	VC	Sa 14-16 06-IV	2	TA 63	C 63	63@m.co
21-320	5	Card	79	79	0	VC	Sa 14-16 06-IV	2	TA 23	C 23	23@m.co
21-320	5	Card	79	79	0	VC	Sa 14-16 06-IV	2	TA 55	C 55	55@m.co
21-314	6	Card	79	79	0	VC	Sa 14-16 06-IV	2	TA 24	C 24	24@m.co
21-314	6	Card	79	79	0	VC	Sa 14-16 06-IV	2	TA 59	C 59	59@m.co
46-210	7	Card	52	52	0	VC	Sa 14-16 06-IV	1	Lec 17	C 1700	Lec17@m.co
21-307	8	Card	52	52	0	VC	Sa 14-16 06-IV	1	Lec 25	C 2500	Lec25@m.co
16-223	9	Card	63	53	10	VC	Sa 14-16 06-IV	2	TA 4	C 4	4@m.co
16-223	9	Card	63	53	10	VC	Sa 14-16 06-IV	2	TA 67	C 67	67@m.co
Supervisor 1	na	na	na	na	na	VC	Sa 14-16 06-IV	1	TA 46	C 46	46@m.co

Table 8: Example of Scheduled Rooms CV, 608 Students

Room	Envelope	Proctors	Observations	Capacity	Students	Slack	Test	Date
46-210	1	1	Card	52	52	0	VC	Sa 14-16 06-IV
21-314	2	2	Card	79	79	0	VC	Sa 14-16 06-IV
16-223	3	2	Card	63	53	10	VC	Sa 14-16 06-IV
46-209	4	1	Card	50	50	0	VC	Sa 14-16 06-IV
46-307	5	2	Card	80	80	0	VC	Sa 14-16 06-IV
21-307	6	1	Card	52	52	0	VC	Sa 14-16 06-IV
16-224	7	2	Card	72	57	15	VC	Sa 14-16 06-IV
21-320	8	2	Card	79	79	0	VC	Sa 14-16 06-IV
41-103	9	2	Doorkeeper	106	106	0	VC	Sa 14-16 06-IV
Supervisor 1	na	1	na	na	na	na	VC	Sa 14-16 06-IV

Table 9: Example of Updated Proctor Log

Name	Cell	email	ID	Experience	Level	ODE, 04-II	AVG, 04-III	Total
TA 1	C 1	1@m.co	ID 1	1	Undergraduate		1	1
TA 2	C 2	2@m.co	ID 2	2	Undergraduate	1		1
TA 3	C 3	3@m.co	ID 3	1	Undergraduate	1	1	2
TA 4	C 4	4@m.co	ID 4	1	Undergraduate	1		1
TA 5	C 5	5@m.co	ID 5	2	Undergraduate	1	1	2
TA 6	C 6	6@m.co	ID 6	2	Postgraduate			0

(i) *Typically, the updated version of the proctors' log should increase its columns in more than one. In our case study, the School of Mathematics from Universidad Nacional de Colombia, Sede Medellín, the update increases seven columns each round because it is the number of massive courses that RaPID⨍Ω manages for the User Institution.*

(ii) *The more courses programmed in one round, the more chances for optimization instances. The examination rounds need not be equal as some courses may take two midterms, while others take three.*

(iii) *The file New_Proctor_Log.xls is created independently from Proctor_Log.xls, instead of simply overwriting it, for security reasons. It will also be useful for later manual corrections, for instance, some employees selected to proctor may have a license (medical or personal). Due to the random nature of these exceptions, it will be wiser to handle them manually, by a human supervisor, than trying to incorporate them in the system.*

(iv) *Once the round of examinations is over, for the next round of tests, the Proctor_Log.xls file must be replaced by the New_Proctor_Log.xls file.*

3.3. The Scheduled_Rooms.xls file

The first module of the system **Room_Decision.py** processes the file Available_Rooms.xls (see Table 2) and chooses rooms in order to minimize the number of necessary proctors (see Section 5 for the exposition of its algorithm). Its results are summarized in the file **Scheduled_Rooms.xls**, which is an excel book having one sheet for each programmed test in the round. Each sheet has the structure of Table 8. As can be observed, it is very similar to the Proposed_Programming.xls file but without the assigned proctors. This is because the Scheduled_Rooms.xls file is an intermediate step, used only after the rooms have been decided. Hence, it is an internal file, moreover, it is part of the input data for the module **Personnel_Decision.py**, which selects a team of proctors (see Section 6 for the presentation of its algorithm).

3.4. The Scheduled_Crew.xls file

The second module of the system **Personnel_Decision.py** processes the input files Personnel_Time.xls, Proctor_Log.xls, Professors.xls together with the internal file Scheduled_Rooms.xls to make job assignment decisions based on fairness. More specifically, the system tries to keep as close as possible, the number of shifts that each TA has in his/her service record. Its decisions are summarized in the file **Scheduled_Crew.xls**, a minimal (and incomplete) example can be observed in Table 10. At this stage, each examination has a crew of proctors which can be gathered/identified by the label **Test**, indicating which test is going to be proctored by each individual in the file. This file, together with Scheduled_Rooms.xls are the input data for the third module, **Crew_Organization.py** used to decide how to organize the previously selected team according to proctoring quality control (see Section 7 for the explanation of the algorithm).

Table 10: Example of Scheduled Crew file

Cell	Experience	Level	Name	Test	email
C 12	3	Undergraduate	TA 12	LA	12@m.co
C 11	1	Undergraduate	TA 11	ODE	11@m.co
C 10	2	Undergraduate	TA 10	DC	10@m.co
C 10	2	Undergraduate	TA 10	IC	10@m.co
C 800	10	PhD	Lec 8	LA	Lec8@m.co
C 700	10	PhD	Lec 7	DC	Lec7@m.co

4. EXECUTION AND PROBLEMS DESCRIPTION

4.1. Execution

In order to run the program, notice that the downloaded folder will contain all the necessary files for a **full example**, which are:

(i) The Python scripts: Room_Decision.py, Personnel_Decision.py, Crew_Organizacion.py, Attendance_Lists.py, and RaPID-Omega.py.
(ii) The input files: Available_Rooms.xls, Room_Data.xls, Personnel_Time.xls, Proctor_Log.xls, and Professors.xls.

The program **runs** from **command line** on a computer having installed Python 3.4.4 or later using the instruction:

python3.4 RaPID-Omega.py -t 54 (54 states the student-proctor rate).

Once the program is executed, the following files are generated: Scheduled_Rooms.xls, Scheduled_Crew.xls, Proctor_Log.xls, New_Proctor_Log.xls, and Proposed_Programming.xls

4.2. Problems Description

The system RaPID$\not{\!\Omega}$ depends critically on the input data basis, therefore, these files are the main source of potential problems, which we describe below.

(i) **Consistency between data basis.** A common source of errors is the consistency of the labels used across the several data sets. For instance, the time windows used for the tests do not agree in format with those of the personnel time availability, or the name of an employee in the data base Personnel_Time.xls does not agree with his/her name in the file Registro_Vigilancias.xls. See the section 2 for more details and examples.

(ii) **Repeated data.** It is possible that files Personnel_Time.xls and/or Registro_Vigilancias.xls present two or more repeated names, due to human error or mere coincidence (two employees with the same name). This will cause errors in the execution of the program. First, make sure that there are no mistakes in the names written in both data basis. If by coincidence two or more employees have the same name, **differentiate them artificially**, e.g., John I, John II, etc.

5. THE MODULE ROOM_Decision.py

The first module of RaPID$\not\Omega$ consists in choosing, for each test, a set of rooms that minimizes the number of needed proctors. This process is done independently for each test in the round because, in most of the cases, the examinations are done in different time windows and therefore, the rooms are not exchangeable items; it consists in two steps. First, the choice of rooms, this is modeled with a 0-1 knapsack problem and solved with dynamic programming algorithm, which is implemented in algorithm 1. Second, the optimal use of the slack between students and available seats (which usually is nonzero), which is solved with a greedy algorithm, implemented in algorithm 2.

5.1. CHOICE OF ROOMS

In this subsection we explain how the rooms are chosen. Before starting, two values need to be introduced

Definition 7. *Let $\big(c_i : 1 \leq i \leq N\big)$ be the list of capacities (quantity of seats) of the available rooms for a given test, define*

$$\begin{aligned} & r \in \mathbb{N} \text{ the student-proctor rate,} \\ & w_i \overset{\textbf{def}}{=} \Big\lceil \frac{c_i}{r} \Big\rceil \text{ the weight/cost of each room.} \end{aligned} \tag{1}$$

Here, it is understood that the label $i = 1, 2, \ldots, N$ stands for each available room and that the ceiling function $x \mapsto \lceil x \rceil \overset{\textbf{def}}{=} \min\{n \in \mathbb{Z} : n \geq x\}$, assigns to any real number $x \in \mathbb{R}$ the minimum integer greater than or equal to x. The student-proctor rate, r, is the number of students that an individual must proctor, the system has ***54 as default value****.*

Therefore, the problem of choosing rooms is modeled by the following binary integer problem

Problem 8. *Let* $\big(c_i : 1 \leq i \leq N\big)$ *be the list of capacities of the available rooms for a given test and let* D *(the demand) be the number of students taking the test. Let* $(w_i : 1 \leq i \leq N)$ *be the list of weights for each room as introduced in Definition 7 then, the problem of minimizing proctors is given by*

$$\min \sum_{i=1}^{N} w_i x_i. \tag{2a}$$

Subject to

$$\sum_{i=1}^{N} c_i x_i \geq D, \qquad x_i \in \{0,1\}, \qquad \text{for all } i = 1, 2, \ldots, N. \tag{2b}$$

Here, for each $i = 1, \ldots, N$*, the binary variable* x_i *indicates whether the room* i *is chosen* $(x_i = 1)$ *or not* $(x_i = 0)$*.*

It is a well-known fact that the solution of Problem 8 above satisfies $x_i \equiv 1 - \xi_i$ for all $i = 1, \ldots, N$, where $(\xi_i)_{i=1}^{N}$ is the solution to the following 0-1 knapsack problem

Problem 9.

$$\max \sum_{i=1}^{N} w_i \xi_i. \tag{3a}$$

Subject to

$$\sum_{i=1}^{N} c_i \xi_i \leq \sum_{i=1}^{N} c_i - D, \qquad \xi_i \in \{0,1\}, \qquad \text{for all } i = 1, \ldots, N. \tag{3b}$$

There are several ways for solving the 0-1 knapsack problem 9. In the RaPID$\not{\Omega}$ system the problem is solved using the technique of dynamic programming (see *Section 11.3* in [1] for details). The implementation is given by algorithm 1

5.2. The Slack between available seats and students

For most instances of Problem 8, the optimal solution $(x_i)_{i=1}^{N}$ will not satisfy actively the constraint (2b), i.e., there will be a slack between the number of available seats $(\sum_{i=1}^{N} c_i x_i)$ and the students taking the test (D). The distribution of the slack among the chosen rooms gives rise to a new optimization process. Define the following parameters

Definition 10. *Let* $\big(c_i : 1 \leq i \leq N\big)$ *be the list of capacities of the available rooms for a given test, let* D *be the number of students taking the test and* r *be the student-proctor rate. Let* $(x_i)_{i=1}^{N}$ *be an optimal solution of Problem 8.*

(i) *We say that the total slack is given by* $S \overset{\textbf{def}}{=} \sum\limits_{i=1}^{N} c_i x_i - D$*.*

Algorithm 1 Room Decision Algorithm, decides the choice of rooms and the slack distribution once an optimal solution to Problem 8 is found.

procedure ROOM DECISION(Available_Rooms.xls file, Student-Proctor Rate: r.
User Decision: Student-Proctor rate r)
  create the Excel book Scheduled_Rooms.xls
  for column of **Available_Rooms.xls do** ▷ Each column is a test, e.g., Table 2
    create the sheet corresponding to the test.
    retrieve from **Available_Rooms.xls** the information: Rooms' List, Capacities: $(c_i)_{i=1}^N$ and Demand: D corresponding to the test.
    call dynamic programming solver (Input: $\{(c_i)_{i=1}^N, \sum_{i=1}^N c_i - D\}$, Output $(\xi_i)_{i=1}^N$)
    compute $x_i \overset{\textbf{def}}{=} \xi_i$ for $i = 1, \ldots, N$
    call Algorithm 2 (Input $\{(c_i)_{i=1}^N, (x_{i=1}^N), D\}$, Output: $(E_i)_{i=1}^N$ quantity of students in each chosen room)
    save $(x_i)_{i=1}^N, (E_i)_{i=1}^N, (w_i)_{i=1}^N$ in the sheet corresponding to the test together with the remaining information displayed in Table 8.
  end for
  save book Scheduled_Rooms.xls
end procedure

(ii) *For each room, define the list of slack priority coefficients by*

$$s_i \equiv c_i \mod r, \qquad \text{with } 0 \leq s_i < r, \tag{4}$$

for all $i = 1, \ldots, N$. Where it is understood that $c_i = q_i r + s_i$ with $0 \leq s_i < r$ being the remainder output in the Euclid's Division Algorithm (see Section 3.3 in [4] for details).

With the definitions above, the greedy algorithm 2 below is implemented.

Remark 11. *Essentially, algorithm 2 tries to optimize the (previously attained solution) slack in order to reduce the number of proctors. For instance, suppose there are only two rooms available with of 55 seats each, 108 students and the student-proctor rate is $r = 54$. Then, the problem 8 will choose both rooms, i.e., $x_1 = x_2 = 1$. Next, the greedy algorithm will program 54 students in each room in order to need 2 proctors, instead of programming 53, 55 which would demand 3 proctors.*

6. THE MODULE PERSONNEL_Decision.py

Once the Scheduled_Rooms.xls file is created, the next step is to choose a proctoring crew for each test, trying to maximize the equity of assigned shifts among

Algorithm 2 Optimal Slack Distribution Algorithm, decides the slack distribution once an optimal solution $(x_i)_{i=1}^N$ to Problem 8 is found.

procedure SLACK DISTRIBUTION(Capacities: $(c_i)_{i=1}^N$, Demand: D, Student-Proctor Rate: r)
    compute list of specific slack priority coefficients $(s_i)_{i=1}^N$ ▷ Introduced in Definition 10.
    sort the list $(s_i)_{i=1}^N$ in ascending order
    denote by $\sigma \in \mathcal{S}[N]$ the associated ordering permutation, i.e.,

$$s_{\sigma(i)} \leq s_{\sigma(i+1)}, \qquad \text{for all } i = 1, \ldots, N-1. \tag{5}$$

    $Slack \overset{\textbf{def}}{=} \sum_{i=1}^N c_i - D$ ▷ Initializing the available slack
    $e_i \overset{\textbf{def}}{=} c_i$ for all $i = 1, \ldots, N$ ▷ Initializing the located students in each room (overcounted)
    for $i = 1, \ldots, N$ **do**
        if $(0 < Slack)$ and $(Slack \leq s_{\sigma(i)})$ **then**
            $e_{\sigma(i)} = e_{\sigma(i)} - Slack$
        else if $(Slack > s_{\sigma(i)})$ **then**
            $e_{\sigma(i)} = e_{\sigma(i)} - s_{\sigma(i)}$
            $Slack = Slack - (s_{\sigma(i)} + 1)$
        end if
        compute $w_i \overset{\textbf{def}}{=} \lceil \frac{e_i}{r} \rceil$. ▷ Updated number of needed proctors
    end for
    return $(e_i)_{i=1}^N, (w_i)_{i=1}^N$
end procedure

TAs, once the lecturers have been directly assigned to proctor in their corresponding subject. Given that all the TAs are exchangeable (unlike the lecturers), the problem of choosing a proctoring crew will be done simultaneously for all the tests in the round, in contrast to the test-wise design of the module Room_Decision.py. The section is divided in two parts, the presentation of the mathematical model, and the presentation of the algorithm.

6.1. The mathematical model

In this section we derive a mathematical model for the problem of selecting personnel for proctoring duties (or shifts). It will be seen that the model is the Job Assignment problem with constraints. Before we introduce it, some definitions are given

Definition 12. *Let T be the number of tests in the round and let P be the total number of part-time employed TAs.*

(i) *For each test t, let $N = N(t)$ be the number of scheduled rooms for the test and let $(w_i^{(t)})_{i=1}^{N(t)}$ be the list of needed proctors in each booked room. Then, the total number of proctors needed in the test is given by*

$$W^{(t)} \overset{\textbf{def}}{=} \sum_{i=1}^{N(t)} w_i^{(t)}. \tag{6}$$

(ii) *For each $t = 1, \ldots, T$ and $p = 1, \ldots, P$, let $y_p^{(t)} \in \{0, 1\}$ be the decision variable defined by*

$$y_p^{(t)} \overset{\textbf{def}}{=} \begin{cases} 1, & \textit{if proctor } p \textit{ is assigned to test } t\,, \\ 0, & \textit{otherwise.} \end{cases} \tag{7}$$

Also define the availability coefficient $a_p^{(t)} \in \{0, 1\}$ by

$$a_p^{(t)} \overset{\textbf{def}}{=} \begin{cases} 1, & \textit{if proctor } p \textit{ is available at the time of test } t, \\ 0, & \textit{otherwise}\ . \end{cases} \tag{8}$$

(iii) *For each p (identifying a TA), denote by L_p the number of served shifts in the Proctor_Log.xls file (i.e., the TA's service record before running the algorithm).*

(iv) *The global service average α, after the current round of tests, is computed in the natural way, i.e.,*

$$\alpha \overset{\textbf{def}}{=} \frac{1}{P}\Big(\sum_{p=1}^{P} L_p + \sum_{t=1}^{T} W^{(t)}\Big). \tag{9}$$

Remark 13. *Observe that due to the expression* (9) *the quantity* α *is known. On the other hand, observe that* $\sum_{t=1}^{T} y_p^{(t)}$ *quantifies the total number of shifts that each TA has in a round of tests. The alternative definition* $\alpha = \frac{1}{P}\sum_{p=1}^{P}(L_p + \sum_{t=1}^{T} y_p^{(t)})$*, would be mathematically equivalent to* (9)*, but it would (unnecessarily) include* α *as a variable.*

With the definitions above, it is direct to see that we want to solve the following problem

Problem 14.

$$\min \max_{p=1}^{P} \Big| \sum_{t=1}^{T} y_p^{(t)} + L_p - \alpha \Big|. \tag{10a}$$

Subject to

$$\sum_{p=1}^{P} a_p^{(t)} y_p^{(t)} = W^{(t)}, \qquad \textit{for all } t = 1, \ldots, T. \tag{10b}$$

$$y_p^{(t)} \in \{0, 1\}, \qquad \textit{for all } t = 1, \ldots T \textit{ and } p = 1, \ldots, P. \tag{10c}$$

Clearly, the problem 14 is not a linear programming problem. In order to transform it, we introduce a new continuous variable z (not necessarily integer), verifying the constraints $\big| \sum_{t=1}^{T} y_p^{(t)} + L_p - \alpha \big| \leq z$ for all $p = 1 \ldots, P$. The absolute sense can be decoupled in two inequality constraints and given that this is an integer programming problem, the following refinement is introduced

$$\begin{aligned} &\sum_{t=1}^{T} y_p^{(t)} + L_p - \lceil \alpha \rceil \leq \sum_{t=1}^{T} y_p^{(t)} + L_p - \alpha \leq z, \\ -z \leq &\sum_{t=1}^{T} y_p^{(t)} + L_p - \alpha \leq \sum_{t=1}^{T} y_p^{(t)} + L_p - \lfloor \alpha \rfloor. \end{aligned} \tag{11}$$

Also recall that for the job assignment problem, the binary integer choice constraint (10c) can be replaced by the natural linear relaxation constraint (12d) below, and the optimal solution of the continuous linear relaxation problem is the optimal solution of the integer problem (see Chapter 4 in [2]). With the introduction of the variable z above and the natural linear relaxation, the problem 14 can be reformulated as the following linear optimization problem.

Problem 15. *With the variables and quantities introduced in Definition 12, the problem of assigning TAs for proctoring duties as fairly as possible, considering their time constraints, is modeled by the problem*

$$\min z. \tag{12a}$$

Algorithm 3 Personnel Decision Algorithm, decides the proctoring crew for all the tests in the round.

```
procedure PERSONNNEL DECISION(Scheduled_Rooms.xls file, Personnel_Time.xls, Proctor_Log.xls, Profesors.xls.)
    load the information corresponding to the input files.
    include the lecturers in the file Scheduled_Crew.xls, assigned to their corresponding courses.
    compute the number of necessary proctors for each test.    ▷ Coefficients (W^(t), t = 1, ..., T), Equation, (6).
    account for the time constraint coefficients for each TA.    ▷ Coefficientes (a_p^(t), t = 1, ..., T, p = 1, ..., P), Equation (8).
    include the proctors necessity constraints    ▷ Equation (12c).
    compute the service average value    ▷ Coefficient α, Equation (9).
    include the proctors service-equity constraints    ▷ Equation (12b).
    solve the Problem 15 ← call a linear program solver    ▷ The system uses scipy.optimize.linprog
    create and save the Updated_Proctor_Log.xls file.
    create and save the Scheduled_Crew.xls file.
end procedure
```

Subject to

$$\begin{aligned} &\sum_{t=1}^{T} y_p^{(t)} - z \leq -L_p + \lceil \alpha \rceil, \\ &\sum_{t=1}^{T} y_p^{(t)} - z \leq L_p - \lfloor \alpha \rfloor, \quad \forall p = 1, \dots, P. \end{aligned} \tag{12b}$$

$$\sum_{p=1}^{P} a_p^{(t)} y_p^{(t)} = W^{(t)}, \qquad \textit{for all } t = 1, \dots, T. \tag{12c}$$

$$0 \leq y_p^{(t)} \leq 1, \qquad \textit{for all } t = 1, \dots, T \textit{ and } p = 1, \dots, P. \tag{12d}$$

6.2. The Personnel_Decision.py module

With the exposition above, the algorithm of the module Personnel_Decision.py is summarized in the pseudocode 3.

7. THE MODULE CREW_Organization.py

Once the Scheduled_Crew.xls file is created, the final step is to distribute the TAs in strategic positions in order to maximize the proctoring quality. More specifically, it is clear that rooms with only one proctor assigned should get more experienced proctors, while a mixed experienced-unexperienced couple should be assigned to rooms needing two proctors. In the same fashion, the supervisor should be an experienced undergraduate TA, as it is a position with higher responsibility. At this point the proctors are no longer exchangeable between tests, therefore the process is done independently as in the module Room_Decision.py. Although it is possible to construct a mathematical integer programming model for this stage, it is more practical to use a greedy algorithm for making these decisions (see [3] for a comprehensive exposition on greedy algorithms). The greedy algorithm 4 is based on lexicographic sorting of the scheduled crew's data. More specifically, a pairing (match) has to be done: the proctoring positions and the proctors themselves. Hence, the proctors will be sorted according to their level in the first place (3 Undergraduate, 2 Graduate, 1 Lecturer) and to their experience in the second place (see Table 4). The proctoring positions need further explanation. Observe that any test will need a number of supervisors (according to the enrollment size) and a number of proctors in each room. The process of selecting supervisors and the process of selecting proctoring positions are different. So, two separate steps will be done for completing the tasks.

With the observations above, the Crew_Organization.py module works with the algorithm 4 below.

Table 11: Example of Scheduled Rooms Remark 16

Room	Proctors	Students
16-223	2	63
46-209	1	50
46-307	2	80
Supervisor 1		

Table 12: Example of Sorted Rooms Remark 16

Room	Proctors	Position	Students
46-209	1	1	50
46-307	2	1	80
16-223	2	1	63
46-307	2	2	80
16-223	2	2	63
Supervisor 1			

Remark 16. *Some clarifications are necessary for a deeper understanding of the design and purpose of the algorithm 4.*

(i) *To understand the line 9 in Algorithm 4 consider the example of Table 11. One supervisor is needed, the room 46-209 needs one proctor while the rooms*

Algorithm 4 Crew Organization Algorithm, decides proctoring positions.

1: **procedure** CREW ORGANIZATION(Scheduled_Rooms.xls, Scheduled_Crew.xls)
2: **load** the sheet Scheduled_Crew.xls as Data Frame
3: **load** the Excel book Scheduled_Rooms.xls as Data Frame
4: **create** column "Num_Level" in Scheduled_Rooms.xls (3 Undergraduate, 2 Graduate, 1 PhD)
5: **create** the Excel book Proposed_Programming.xls
6: **for** test_idx in **Scheduled_Rooms.xls do** ▷ Each sheet in the book is a test.
7: **create** sheet test_idx in the book Proposed_Programming.xls
8: **select** the rooms scheduled for test_idx, i.e., Scheduled_Rooms[test = test_idx].
9: **create** the Local_Test_Frame, with columns: Room, Proctors, Position, Students ▷ The column "Position" indicates if it is the 1st, 2nd, 3rd... proctor in the room.
10: **define** $ns \stackrel{\textbf{def}}{=}$ the number of needed supervisors.
11: **create** Local_Proctors_Frame ← Scheduled_Crew[Test = test_idx] ▷ The proctors assigned to the test.
12: **sort** lexicographically the Local_Proctors_Frame with the priority: ["Num_Level", "Experience"] in the order [Descending, Descending].
13: **select** Undergraduate_Local_Proctors_Frame ← Local_Proctors_Frame[Level = Undergraduate] ▷ Choose the undergraduate proctors from the local scheduled proctors frame.
14: **assign** the first ns proctors from Undergraduate_Local_Proctors_Frame as supervisors.
15: **save** the supervisors in the sheet test_idx
16: **remove** the chosen supervisors from Local_Proctors_Frame.
17: **remove** the supervisors' rows from Local_Test_Frame.
18: **sort** lexicographically the Local_Test_Frame with the priority: ["Proctors", "Position", "Students"] in the orders [Ascending, Ascending, Descending].
19: **paste** Local_Test_Frame with Local_Test_Frame and **define** the outcome as Local_Programming.
20: **save** Local_Programming in the sheet test_idx of the book Proposed_Programming.xls.
21: **end for**
22: **save** book Proposed_Programming.xls
23: **end procedure**

16-223 and 46-307 need two. Hence, a data frame including positions, has to be created; see Table 12.

(ii) *The command of line 18 in Algorithm 4 has the following motivation. Con-*

sider the example of Table 11. Then, line 16 in Algorithm 4 would deliver table 12 below. It is clear that once the scheduled proctors are sorted as indicated in line 10 of algorithm 4, the most experienced proctors will be distributed, as evenly as possible, through the rooms; taking the rows with label "1" in the column "Position". Next, the less experienced proctors will be assigned to the rows having "2" in the column "Position", and so forth.

Acknowledgements: The Author wishes to thank Universidad Nacional de Colombia, Sede Medellín for supporting the production of this work through the project Hermes 45713.

REFERENCES

[1] Bertsimas, D., and Tritsiklis, J., N., *Introduction to Linear Optimization*, Athena Scientific and Dynamic Ideas, LLC, Belmont, MA, 1997.
[2] Conforti, M., Cornuéjlos, G., and Zambelli, G., *Integer Programming*, Graduate Texts in Mathematics, Springer, Switzerland, 2014.
[3] Dasgupta, S., Papadimitriou, C., and Vazirani, U., *Algorithms*, Education, McGraw-Hill, New York, NY, first edition, 2006.
[4] Johnsonbaugh, R., *Discrete Mathematics*, Prentice Hall, Upper Saddle River, NJ, 5th edition, 2001.