ULg Library

http://lib.ulg.ac.be

http://matheo.ulg.ac.be

Master thesis : Series partitioning for exam scheduling

Auteur : Morelle, Clément
Promoteur(s) : Louveaux, Quentin
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil en informatique, à finalité spécialisée en "intelligent systems"
Année académique : 2016-2017
URI/URL : http://hdl.handle.net/2268.2/3200

Avertissement à l'attention des usagers :

Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.

Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.

LIÈGE université

Faculté de Sciences appliquées

Master Thesis

Series partitioning for exam scheduling

Clément Morelle

Mentor
Quentin Louveaux

August 20, 2017

Contents

1 Introduction

In the Belgian higher education system, the academic year is organized in the following way : a first lecturing period extends from September to December and a second period from February to Mai. The courses are held within one of these two periods. Each lecturing period is followed by an examination period (3 weeks) during which (almost) all the courses are evaluated. The evaluation period is short and need to be organized in such a way that each exam can take place, that no teacher has to perform two exams at the same time and that no student has two exams at the same time.

Until 2014, date on which an educational reform has been applied in the "communauté française de belgique" (french speaking community of Belgium), the curicula were relatively simple to organize. A student had to succeed (almost) all the courses of his curriculum to be granted an access to the next grade in his studies. Consequently most of the students of a given grade and a given option had the same program. Scheduling the exams was therefore relatively simple. Indeed, the groups of courses corresponding to each grade were quite independent from each other in terms of students, and thus reasoning about schedule could be done, more or less each grade separately without introducing many conflicts in the final exam schedule (assuming no limitation due to lack of classrooms). The curriculum of a student, also called his program of courses, refers to the set of courses that a student must follow and for which he is evaluated at the end of each lecturing period.

In 2015, the educational reform introduced some changes in the academic organization. Among other things it brought more flexibility in the student's programs. In the previous system, student's cursus were divided into 3 to 5 distinct grades of study (5 for universities). Each grade had its set of courses. With the reform this division disappears and is replaced by a system with prerequisites. Each course can be chosen at the condition of having succeeded a list of required courses.

Since this reform, making schedule has become a giant puzzle. The first reason is the disappearance of the obligation of succeeding all the courses of a given grade to have access to the courses of the following one. Therefore student's programs are more diverse and the students following a given course may be scattered over several grades. Altough there is no official notion of "grade" anymore, I refer to it as the year of study for a "normal" student who succeeds all of his courses. Moreover the reform has allowed the access of master grades to students coming from other subject. Some of them must do a preliminary "master foundation year" ("passerelle" in french) year in order to have the required background for their new subject. Those students have a custom program which implies also more diversity. Other — this is my case — are granted the access to the master despite a different bachelor. They have a different program than the regular students during the 2 years of masters. All this explains why setting-up practicable exam schedules over ony 3 weeks has become a real challenge.

Given the present diversity of student programs, even finding a compatible solution were no student has two exams simultaneously is not a trivial task. In addition, the teachers must be available as well as classrooms (limited in number and size). It becomes even more complex when we try to find a schedule that optimizes some measurable quality features. A high quality schedule in the student perspective is a schedule where the exams are well spaced out and were the time before an exam depends on the difficulty of that exam. Another source of complexity comes from the fact that some courses are evaluated with oral exams. This imposes splitting the students attending those courses into smaller oral groups. Indeed evaluating students orally and thus individually takes time and during a period of time usually allocated to an exam (4 hours) the teacher can only evaluate a small group of students, usually around 12. As most courses are followed by more than 12 students, they must be partitioned across several periods of time. Those groups of student evaluated during a single period of time are called "series". This thesis will address the problem of partitioning the student for oral evaluation. But before defining the objective of the thesis in more details let us define some vocabulary : the words *course*, *exam*, *student* and *serie*. They are important notions of the problem therefore it is important to define precisely their meaning in this context.

Course : A *course* is a subject given by one or several teacher(s) over one semester. A *course* is followed by one or more *students*. A *student* is a person following one or several *courses*. A *student* must attend the

examination(s) of all of his/her *courses*. The evaluation of a *course* is either oral or written. In real situation, some courses have an oral and a written examination but in that case, the pre-processing splits them into 2 distinct courses. The courses that are not evaluated are ignored from the scheduling problem.

Exam : An *exam* is the evaluation of a group of students on the subject of a course. An *exam* is associated to exactly one *course*. An *exam* occurs at a given time during the session. An *exam* represents both oral and written evaluations. The group of students of an *exam* is a subset of the students following the corresponding *course*. A course may have several exams (in the case of course evaluated orally).

Series : A *series* is an oral *exam*.

The relation *course-exam* is a "one-to-many" relation. A course can be associated to one or several *exams* but an *exam* is associated with only one *course*. Any *student* of a *course* must be in exactly one of the *exams* associated to that *course*. In other words, the *students* of a *course* are split into one or more *exams*. If a *course* has a written evaluation it is associated to only one *exam* containing all the student. If a *course* has a oral evaluation it is associated to one or more *series* depending on the maximum size of the series for that course.

The problem of finding the optimal schedule is complex, probably NP-hard, and it is hard to think of an algorithm that solves it using imperative programming. Luckily there are other tools that allow expressing the problem in an exact formulation like *Mathematical Programming*. Using this tool, we could try to solve the whole problem at once, expressing all constraints and all relations between variables as well as a score function to be minimized/maximized. The advantage of this technique is its exactness but the downside of the method is that it can take a very long time to get to this optimum. The problem of exam scheduling is so complex that solving it using MILP model can not be done in realistic time. A compromise must be found between exactness and computing time, which leads us to alternative approaches.

The problem being too complex to be solved as a whole, we must try to decompose it in a way that simplifies its sub-parts but does not affect too much the quality of the final result. In this perspective a good idea would be to create the series for the oral courses prior to the computing of the schedule. Indeed it simplifies a lot the model for the schedule generation. If all series have been defined in a first time, we do not need the notion of individual student for the later problem. We must only know which pair of exams can or cannot occur at the same time. It reduces drastically the number of variables. This thesis is based on this idea : the division of the problem of exam scheduling in, on one hand, the series partitioning and, on the other hand, the schedule making. The objective will be to find a method solving the first problem (series partitioning) and to discuss the quality of the final solution : how does it work when we bring together both model to perform the final schedule?

This approach creates the problem of partitioning some courses into series. It can be stated as follows : What is the best strategy to split the students from a course into several series for oral evaluation in such a way that the final schedule generation is the best (according to some objective function)? This will be the main problem of this thesis. Which criteria should we base on to solve this problem given that we can not directly measure the relation between a given split and the quality of the final schedule. Intuitively, we feel that students who have the similar program should be put in the same oral series.

To illustrate this intuition let us take a simplistic example where we have two distinct groups of students (red and green on Figure 1), corresponding for example to two different curriculum. Each group has one course on their own : course A for the red group and course B for the green group. They have also one course together : course C. We assume that course C has to be split into two series for the exams. In a first scenario we split the course C randomly so that each serie (C1,C2) contains students from the two groups (bottom left on Figure 1). In a second scenario we split C accordingly with the groups (bottom right on Figure 1). The two series can not occur at the same time as there is only one teacher evaluating. Let us define as objective to be minimized the number of time periods needed to organize the session. In the first case we need 3 time periods to organize the 4 exams : [A,B],[C1],[C2]. In the second case, we need only 2 time periods [A,C2], [B,C1].

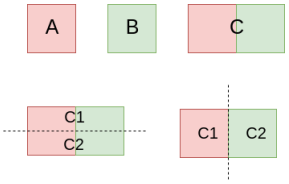


Figure 1: serie assignement simple example

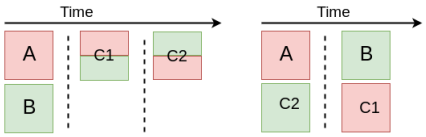


Figure 2: schedule simple example

Here is an instance of this example for explicit students. We have 4 students (1,2,3,4) and three courses (A,B,C). The relation student-course is the following:

	A	B	C
1	x		x
2	x		x
3		x	x
4		x	x

In terms of student/exam our two scenari could be the following :

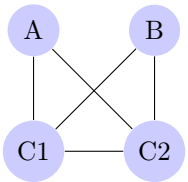
First scenario

	A	B	C1	C2
1	x		x	
2	x			x
3		x	x	
4		x		x

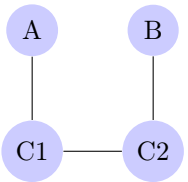
Second scenario

	A	B	C1	C2
1	x		x	
2	x		x	
3		x		x
4		x		x

Let us now analyze the example under the angle of exam incompatibilities. We have 4 exams (A,B,C1,C2). A pair of exam is incompatible if they can not be in the same time slot in the schedule. Either because they are from the same course or because they share common students.



(a) first scenario



(b) second scenario

Figure 3: Simple example - incompatible pairs

The first scenario is worse because there are more pairs of incompatible exams. Each such pair add a constraint on the schedule. In the second senario there are only 3 incompatible pairs. Therefore scenario 2 has more flexibility to arrange the exams in order to achieve some objective, in this case having the shortest exam session.

We see that grouping students with other students having similar programs seems a good idea. There are several ways to implement a model that achieves this.

Quadratic assignment We could try to translate directly our intuition of grouping similar student together mathematically. Let us define a similarity score on each pair of student. This score is equal to the number of courses they have in common. The more similar two students are, the higher their score is. Imagine that we have a feasible solution. We have k exams. Each exam contains a set of students. We can compute the score of one exam as the sum of the similarity scores of all pairs of students in that exam. To get the total score we sum this score for all exam. This score must be maximized. It is expressed mathematically as:

$s_{i,j}$: Student similarity : The number of common courses between the students i and j
$x_{i,k}$: 1 if student i is in serie k, 0 otherwise

The solution of the quadratic assignment problem is the allocation $x_{i,k}$ that maximizes the expression

$$score = \sum_k \sum_{i<j} s_{i,j}\ x_{i,k}\ x_{j,k} \tag{1.1}$$

Quadratic assignment is known to be a hard problem in theory as well as in practice.

Moreover there is a flaw in this approach. Let us imagine a case where all student follow exactly the same program, all pairs of students have the same similarity, which is equal to the total number of courses. If we look back at the equation 1.1 and consider $s_{i,j}$ as a constant S, we get :

$$score = S \sum_k \sum_{i<j} x_{i,k}\ x_{j,k} \tag{1.2}$$

and given that

$$\sum_{i<j} x_{i,k}\ x_{j,k} = \binom{size(k)}{2} \tag{1.3}$$

where

$$size(k) = \sum_l x_{l,k} \tag{1.4}$$

is the number of students in series k.

We get at the end

$$score = S \sum_k \binom{size(k)}{2} \tag{1.5}$$

But we also know that the problem bounds the size of each series. This means that between several partitioning of various qualities, what will change is not really the size of each series but rather their composition. The issue is that in our specific case ($s_{i,j}$ constant) the score is almost not influenced by the way we split each course into series. Knowing that different ways of partitioning can lead to huge differences in term of quality, we can conclude that this score is not correlated to the quality of partitioning and is therefore a bad choice.

Here is an example of this issue. We have 2 courses A and B and 4 students as shown in the table below.

	A	B
1	x	x
2	x	x
3	x	x
4	x	x

Lets assume that both courses A and B have to be split in 2 series of 2 students. We have here two solutions.

$$solution1 = \begin{cases} A1 = \{1, 2\} \\ A2 = \{3, 4\} \\ B1 = \{1, 2\} \\ B2 = \{3, 4\} \end{cases} \tag{1.6}$$

$$solution2 = \begin{cases} A1 = \{1, 2\} \\ A2 = \{3, 4\} \\ B1 = \{1, 3\} \\ B2 = \{2, 4\} \end{cases} \tag{1.7}$$

Both solutions have the same score if we consider quadratic assignment : 16

but if we represent the two solutions as a graph showing the incompatible pairs of exams, it is very clear that the first solution will be of higher quality:

(a) solution 1 (b) solution 2

Figure 4: quadratic assignment - incompatible pairs

We see that we have to find another criteria to measure the quality of a split.

Course union Another way to put our intuition into equation is the following. For each exam i, we look at the number of other exams having students in common with i. This number is thus equal to the cardinality of the union of the sets of exams corresponding to each student from i. We will try to split the courses such that the sum of those number over all the exams is minimized.

In this formulation we do not directly group similar students together but we make such that in average the students of one exam are scattered into a small number of other exams. This could be expressed mathemically as:

$$\begin{gathered} \text{i : students indexes} \\ \text{k : exams indexes} \\ exams\_of\_student(i) \text{ : the set of exams of student i} \\ student\_of\_exam(k) \text{ : the set of students of exam k} \end{gathered}$$

$$score = \sum_{k} \#\Big(\bigcup_{i \in student\_of\_exam(k)} exams\_of\_student(i) \Big) \tag{1.8}$$

Pairs of exam A third way to evaluate the quality of the partitioning would be considering all the pairs between exams and try to minimize that number. It is more or less what we represent on Figure 3 in the case of the simplistic example.

$$score = \min_{partitioning} \sum_{i<j} \Big(notEmpty\big(exam_i \bigcap exam_j \big) \Big) \tag{1.9}$$

where $notEmpty(\cdot)$ is equal to 1 if the ensemble is not empty, 0 otherwise.

If we look back at the simplistic example, we saw that grouping students by similarity induced a lower number of incompatible pairs. In the present formulation we try to act directly on the consequence and not the cause which may be a relevant choice.

This formulation is actually equivalent to the previous one (course union). Indeed, both formulation are counting pairs of exams having common students. The only difference is that in the first formulation, we count every pair twice (one time from each pair-element perspective).

Up to now, we have decided to decompose the whole scheduling problem into two smaller problems and we have proposed different ideas to deal with the problem of series partitioning. Now that we have defined the scores (to be minimized) as a measure of the quality of a solution for the series partitioning sub-problem, let us try to evaluate its complexity.

To have an idea of the combinatorial complexity of the problem, we can calculate the number of possible splits of the courses into series. let us assume we have 100 courses by semester in the faculty (less than in reality), that classes are composed of only 20 students, that one third of the exams are orals and that teachers take students by group of 10. How many possible split is there? Here we are under-estimating the complexity to have an idea of a lower bound.

for one course the number of split is :

$$\frac{\binom{20}{10}}{2!} = 92378 \tag{1.10}$$

So if we have this for 33 courses, we obtain:

$$(92378)^{33} \approx 1.31 \quad 10^{148} \tag{1.11}$$

A brute force approach is thus impossible. Even if we could test a billion configurations per second, it would take $2\ 10^{78}$ years to find the optimum. This gives also an idea of the complexity of the problem. Knowing that in bachelor some courses have more than 100 students the real value is even bigger. Note that for a single class of 100 students divided into 10 groups, the number of split is :

$$\left(\binom{100}{10} \cdot \binom{90}{10} \ldots \binom{20}{10}\right)/10! \approx 6.4 \quad 10^{85} \tag{1.12}$$

2 Model

2.1 Structure of the data

Let us have a closer look at our data to understand its structure and try to get a good intuitive idea of the connectivity between its components. The data consists of two sets of objects — a set A of courses and a set B of students — and of a set P of pairs associating objects from each set. P is thus a subset of the cartesian product $A \times B$ which is the set of all ordered pairs (x,y) such that $x \in A$ and $y \in Y$. The semantic of a pair (x, y) is that the student y ($\in B$) follows the course x ($\in A$).

Each course has some 2 useful meta-information : The type of exam evaluation (oral or written) and in the case of oral evaluation, the maximum number of students per serie. In order to abstract the notion of type of course and make the mathematical expression of our model simpler, we will translate those 2 meta-information into two values :

- The number of exams (1 for written course, 1 or more for oral courses)
- The maximum number of students per exam (infinite for written exam)

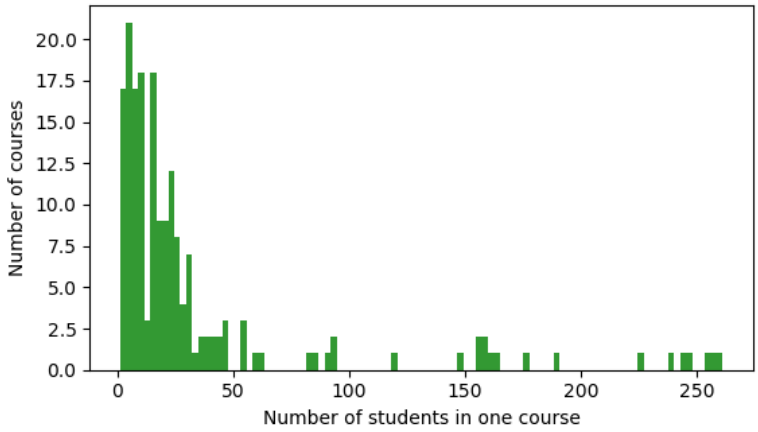


Figure 5: Number of courses having a given number students

From this point we do not need anymore the notion of oral and written course. This simplifies the mathematical expression. All other course's meta-information are meaningless for our problem. They include the name of the courses and teachers and the teachers availability. Students have no meaningful information for our problem.

Let us now try to get an intuition of the data by computing some meaningful values that express how much the courses and the students are intricate/independent. We have at our disposal the data of the faculty of applied sciences of the university of Liège for the exam sessions of January 2016 and June 2016. We will analyze the data of January 2016 but we can assume that the data of June has the same kind of structure.

- Number of courses : 183
- Number of students : 1093
- Number of pairs course-student : 6453

As a comparison, the Cartesian product of the ensemble of courses and student has 200019 pairs. The number of pairs represent thus 3.2% of the total possible pairs students-course. If we had to represent our data as a graph with edges between courses and students, the graph would not actually be very dense. But we do not expect the 3.2% of pairs to be distributed randomly across the Cartesian product ensemble. We expect correlations between subgroups of students and courses.

Now let us observe some statistics about the distribution of the number of students per course and the distribution of the number of courses per students. From Figure 5 and 6, we can see that there is a big variability in the size of the courses (from 1 to 261). Indeed, we know that in the first 2 grades of studies, courses are very general and followed by almost all students. Later in the studies, students choose a specialization and are scattered in smaller groups.

Let us now focus on the pairs between courses. There are $\binom{183}{2} = 16653$ such pairs and among them only 1561 have at least one common student (9%). Independences can thus be found between sub-parts of the problem. One of our solving method will explicitly exploit those independencies.

Let us now express the problem in the *Mathematical Programming* formalism i.e. in terms of variables, constraints and objective to be optimized.

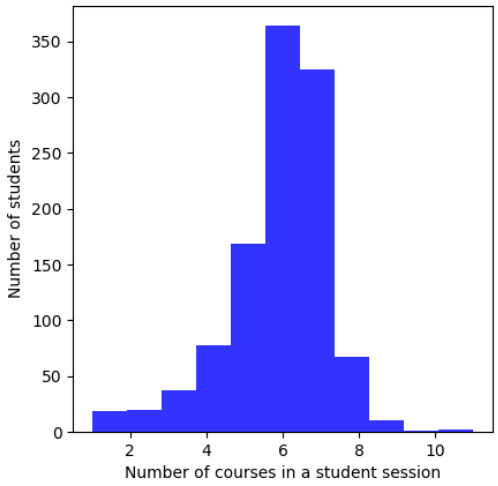


Figure 6: Number of students having a given number of courses in their exam session

2.2 Definition of the model

Data

$$courses\_student_{c,s} = \begin{cases} 1, & \text{if student } s \text{ follows course } c \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

$$courses\_exam_{c,x} = \begin{cases} 1, & \text{if exam } x \text{ is associated with course } c \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

$$capacity_x = \begin{cases} max > 0 (\in N), & \text{if exam } x \text{ has a maximal capacity of } max \\ -1, & \text{if exam } x \text{ is not bounded} \end{cases} \tag{2.3}$$

Variables

$$alloc_{s,x} = \begin{cases} 1, & \text{if student } s \text{ has exam } x \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

$$pair\_student_{x,y,s} = \begin{cases} 1, & \text{if student } s \text{ has both exam } x \text{ and exam } y \\ 0, & \text{otherwise} \end{cases} \tag{2.5}$$

$$pair_{x,y} = \begin{cases} 1, & \text{if exam } x \text{ and exam } y \text{ have at least one student in common} \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

Constraints

Define the variable $pair\_student_{x,y,s}$

$$pair\_student_{x,y,s} = alloc_{s,x} \,.\, alloc_{s,y} \quad \forall x, y, s \tag{2.7}$$

This constraint is not linear. It can not be given to a MILP solver. It has to be translated into a set of linear constraints using the fact that in binary algebra:

$$x = a \,.\, b \Leftrightarrow \begin{cases} x \leq a \\ x \leq b \\ x \geq a + b - 1 \end{cases} \tag{2.8}$$

constraint 2.7 is thus transformed into

$$pair\_student_{x,y,s} \leq alloc_{s,x} \quad \forall x, y, s \tag{2.9}$$

$$pair\_student_{x,y,s} \leq alloc_{s,y} \quad \forall x, y, s \tag{2.10}$$

$$pair\_student_{x,y,s} \geq alloc_{s,x} + alloc_{s,y} - 1 \quad \forall x, y, s \tag{2.11}$$

If a student follows a course he must be in exactly one exam of that course:

$$courses\_student_{c,s} = \sum_{x} courses\_exam_{c,x} \,.\, alloc_{s,x} \quad \forall s, c \tag{2.12}$$

Make sure that the capacity of each serie is not exceeded:

$$\sum_{s} alloc_{s,x} \leq capacity_x \quad \forall x$$

Objective

$$\min \sum_{x,y} pair_{x,y}$$

2.3 Mathematical Integer Linear Programming

There exists solvers able to solve models defined in this formalism. The one used for this thesis is CPLEX. Those solvers are powerful tools: We only need to express the model correctly and we have the guarantee that the solver will converge toward the optimal solution; or if there is no solution the model will be declared unfeasible. We do not need any insight into the complexity of the problem nor give any hint on a way to tackle down the problem. Those solvers use the "branch and bound" algorithm to explore the space of solutions. the space of solutions is represented in a tree structure and branches can be pruned if they correspond to unfeasible or sub-optimal solutions. Those tools use also linear relaxation to get more quickly to the optimum integer solution. Linear relaxation means removing the constraints that integer variables have integer values. It allows using efficient tools from linear algebra.

On the other hand we have very few control on the way the solver proceeds. Our control relies only on the way we define our model i.e. the choice of the data, the variables, the constraints (There may be several formulations of the same problem). Performances can vary from one formulation to another but they also depend on the data.

Even with the most efficient formulation, those solving methods have the drawbacks of their advantages: In order to be exact, they must explore the whole space of solution. however, if the model has a good linear relaxation, exploring the space is sped up. As the complexity of the data grows, those methods reach limits in terms of execution time and/or memory usage.

2.4 Naive MILP implementation

As a first try, let us solve the model given in section 2.2 using a MILP solver. Varying the size of the sample we observe that from a sample size corresponding to 20% of the data available, the model starts to slow down; and by raising it a little bit we soon fill the whole memory of the computer.

This shows that a MILP can work but at the condition of either an improved formulation or of limiting the model to smaller samples.

2.5 MILP improved implementation

Let us improve the MILP formulation in order to solve the out-of-memory issue of the first model. The problem of memory (and speed) of the first model comes mostly from the variables $pair\_student_{x,y,s}$. When the model is applied on a whole exam session, the number of variable is 232*232*1100 which is around 6 10^7.

In that model we create a variable for every combination of exams of any course. But in practice we observe that some courses have no student in common. As a consequence, there are no common students between any of their series. Using that observation and given the data, we can reduce the 232*232 pairs of exams to only a few hundred. We only take into account pairs that could potentially (for some some partitioning) have common students. The computation of the useful pairs is a little pre-processing prior to the definition of the model.

Here is the improved model.

Data

$$courses\_student_{c,s} = \begin{cases} 1, & \text{if student } s \text{ follows course } c \\ 0, & \text{otherwise} \end{cases} \tag{2.13}$$

$$courses\_exam_{c,x} = \begin{cases} 1, & \text{if exam } x \text{ is associated with course } c \\ 0, & \text{otherwise} \end{cases} \tag{2.14}$$

$$capacity_x = \begin{cases} max > 0 (\in N), & \text{if exam } x \text{ has a maximal capacity of } max \\ -1, & \text{if exam } x \text{ is not bounded} \end{cases} \tag{2.15}$$

$$exam\_pairs_p = \{x, y\}, \quad x \text{ and } y \text{ being the index of the exams of pair } p \tag{2.16}$$

Variables

$$alloc_{s,x} = \begin{cases} 1, & \text{if student } s \text{ has exam } x \\ 0, & \text{otherwise} \end{cases} \tag{2.17}$$

$$pair\_student_{p,s} = \begin{cases} 1, & \text{if student } s \text{ has the two exams of pair } p \\ 0, & \text{otherwise} \end{cases} \tag{2.18}$$

$$pair_p = \begin{cases} 1, & \text{if the two exams of pair} p \text{ have at least one student in common} \\ 0, & \text{otherwise} \end{cases} \tag{2.19}$$

Constraints

Define the variable $pair\_student_{x,y,s}$

$$pair\_student_{x,y,s} = alloc_{s,x} \,.\, alloc_{s,y} \quad \forall x, y, s$$

Again this constraint is linearized as :

$$pair\_student_{p,s} \leq alloc_{s,x} \quad \forall p, s \tag{2.20}$$

$$pair\_student_{p,s} \leq alloc_{s,y} \quad \forall p, s \tag{2.21}$$

$$pair\_student_{p,s} \geq alloc_{s,x} + alloc_{s,y} - 1 \quad \forall p, s \tag{2.22}$$

Note that $\{x, y\} = exam\_pairs_p$

If a student follows a course he must be in exactly one exam of that course:

$$courses\_student_{c,s} = \sum_x courses\_exam_{c,x} \,.\, alloc_{s,x} \quad \forall s, c$$

Make sure that the capacity of each serie is not exceeded:

$$\sum_s alloc_{s,x} \leq capacity_x \quad \forall x$$

Objective

$$\min \sum_p pair_p$$

This solves indeed the memory issue and even seems to improve slightly the performances. But from a sample corresponding to 30% of the data it still gets really slow : 15 hours were not enough to solve it. So for solving 100% of the data in one model it would probably be either impossible or extremely long.

2.6 Third MILP implementation

In order to still improve the performances from the second model we could try to optimize the formulation. The idea of the improvement is the following : in the equations 2.20 to 2.22, for a pair of exams we put the constraint on all the students. But the only students that might influence the fact that the pair becomes incompatible are the student that are common to both exams. Using this trick we can reduce the number of constraints.

But doing this modification only removes useless constraints. The result is that for a sample of 20% of the data it takes significantly more time than the second model. By trying to optimize the model, this modification actually weakens it.

2.7 Conclusion about model

The best — or least inefficient — MILP model is the improved MILP formulation from section 2.5. Given the complexity of the problem, it is impossible to find the exact (optimal) solution for realistic instances (the whole data set of one exam session). Therefore, we have to think of different strategies to reduce the complexity of the solving without affecting too much the quality of the solution.

We can conclude by saying that MILP are interesting to solve optimally small problems but they have their limitations in terms of performances. In our case the problem is too complex or not enough linearly "relaxable" to be solved by a MILP solver.

3 Clustering + MILP solving method

We have seen that the frontal approach of expressing and solving the problem as one MILP model does not work in realistic cases. Thus we have to think about an alternative approach. In a first time let us keep using MILP model. Indeed, even if this technique has proven itself unable to solve the whole problem it is still convenient to express complex problems like ours. Moreover we know that when the problem is small enough it is able to find the optimal solution relatively quickly. We want to find a way to decompose our large problem into smaller problem in a clever way and use MILP solver on smaller sub-problems.

Here come the idea of the clustering. If we perform a clustering on the data-set we can decompose it into smaller, easier to solve, sets in a way that keeps together in the same clusters the highly dependent samples and separates the sample that have few in common. The idea is to apply the clustering on the courses and group similar courses together; then do the partitioning of the courses belonging to one of these cluster. This partitioning is done using the MILP model described earlier. Even if we partition only the courses of one cluster, we still take into account all the other courses. The key point to reducing the complexity of the MILP is to avoid partitioning too much courses at the same time. Taking other courses into account (without partitioning them) is not so expensive for the model.

The whole challenge of the clustering is to decompose the problem into sub-problems whose MILP formulation is easy enough to solve but also to decompose in a way that degrades the global score the least possible with respect to the exact solution. Finding the best way of clustering is thus a matter of compromise: decomposing into few large clusters will lead to a more accurate solution but will also take more time as the sub-problems will be difficult to solve. On the other hand a high decomposition will lead to fast solving but will also degrade the final score as each sub-problem will be solved not taking into account other parts of the problem that might influence it. As we know the sum of the individual interests does rarely lead to the general interest.

Clustering is based on the notion of distance between the samples. It tends to group together the samples that are close. But the choice of a distance measure is not so obvious. It is not as simple as in the case of grouping points from a 3D space where an Euclidian distance is the obvious appropriate choice. In our case we want a distance measure that tends to make similar courses close. We base the distance on the number of common students between 2 courses but also on the number of distinct students and thus on the proportion of common students. Indeed if two courses have the exact 10 same students we can feel that they are more similar that two courses of 50 students having 10 students in common.

If our clustering works well, we expect it to group courses more or less by curriculum and by year of study. Why then bother to make a general method if we could simply rely on the knowledge we have of the programs of courses? The first reason is precisely to keep it general. The method should be able to apply to other partitioning problems having a similar formalization. It should also be able to apply to scheduling problem where we do not have the extra knowledge of the curriculum and year of study. The other reason is that given the diversity of the programs, partitioning with respect to year of study or option is probably not optimal anymore and certainly not robust. The whole purpose of the thesis is avoiding to rely on problem-specific knowledge and finding robust method that deduce themselves the structure of the data and adapt to it.

The clustering can thus be seen as a pre-processing to the MILP solving. This is a fast algorithm with respect to MILP solver. The difficulty resides in the choice of the distance measure between the courses as well as finding the optimal level of clustering somewhere between one big group (no splitting) and one group per element (no gathering).

3.1 Choice of a clustering algorithm

There are two main categories of clustering techniques. A first where the number of clusters is predefined (e.g. K-mean method). A second where the method decides itself the number of clusters depending on the data. In our problem, we cannot predict the nature of the data so it would be a bad idea to make an assumption on the number of clusters. We use thus the second category. More precisely we use “hierarchical clustering”.

The principle of hierarchical clustering is the following : it starts with all samples being their own cluster. Then incrementally, it will merge the closest clusters together. While merging clusters, it keeps track in a tree structure of all the merges that are performed until we have only one cluster left.

Let us take a simple example to illustrate the functioning of the method. Let us define some points in the 2D space. They are scattered around three points but with some variance. Figure 7 shows 4 different levels of aggregation showing respectively 12, 6, 3 and 1 cluster. One can see in the third graph (3 clusters) of Figure 7, that the clusters match the points around each center (red cross). So the clustering is able to recover the

categories of points as they were generated. Figure 8 shows the scores of each cluster as they are being built. It begins at the level zero when each point is its own cluster. When two clusters are aggregated the score increase depends on the distance between the two clusters. As we see on Figure 8 the biggest increase in score corresponds to agglomerating the three clusters into a big one. This suggests that the best clustering is 3.

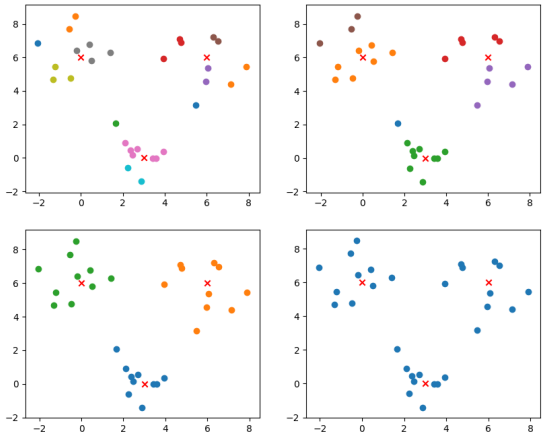


Figure 7: demo 2-D clustering. 12,6,3,1 clusters

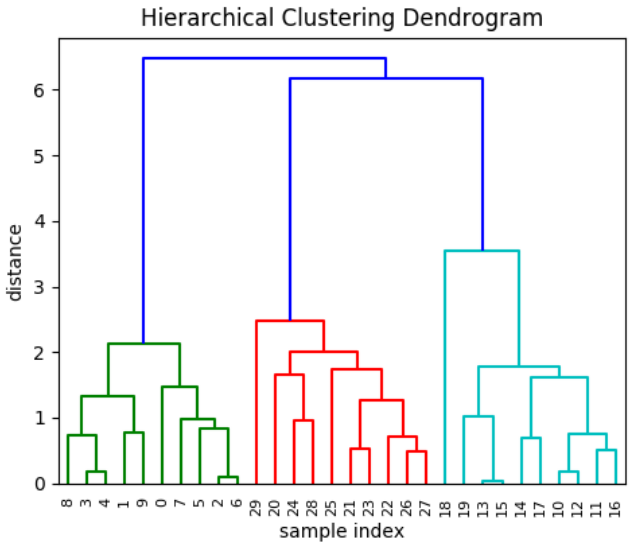


Figure 8: demo 2-D clustering. dendrograme

3.2 Parameter of hierarchical clustering

Distance measure

Defining an appropriate distance measure is fundamental in a clustering problem. It is the measure that will make the courses regroup in one way or another. As I said, we want the distance to be small for similar courses and large for course having few or nothing in common. In order to get a minimum score to our optimization problem

we must try to group students in similar series across different courses. Doing so leads to less incompatible pairs. We must thus give a low distance to courses sharing a lot of similar students. The similarity will thus be computed with respect to the sets of students following the course. For a given pair of courses and for the corresponding sets of students, we can compute the intersection of those sets to get the common students. We can compute the union of those sets to get the total number of involved student. Finally we can also compute the difference between the union and the intersection to get the number of student that are not common to the 2 courses. From those quantities, we can compute other meaningful values like the proportion of common students.

$$union = \#\big(course\_1 \cup course\_2\big) \tag{3.1}$$

$$inter = \#\big(course\_1 \cap course\_2\big) \tag{3.2}$$

$$diff = \#\big(union \setminus inter\big) \tag{3.3}$$

From this data let us define some distances measure. As a first try we could simply take the number of common students:

$$measure_1 = inter^{-1} \tag{3.4}$$

Here we must be careful that *inter* may be equal to zero. But in hierarchical clustering an inter-object distance should never be infinite. To solve this problem, we may think of two solutions. We could simply give a high value (for example one million) to courses sharing no students. We could also solve the problem by adding a small ϵ to the *inter* value, fixing by the way the maximal distance.

$$measure_1 = (inter + \epsilon)^{-1} \tag{3.5}$$

We could also decide to focus more on the proportion of common student rather than on the number:

$$measure_2 = \left(\frac{inter + \epsilon}{union}\right)^{-1} \tag{3.6}$$

We could also combine by multiplying the two first measures in order to take into account both the number and the proportion of common students.

$$measure_3 = \left(\frac{inter^2 + \epsilon}{union}\right)^{-1} \tag{3.7}$$

We could also try to take into account those parameters in a more general non linear way:

$$measure_4 = \left(\frac{inter^a + \epsilon}{union^b}\right)^{-1} \tag{3.8}$$

making a and b vary in some range of real values.

We could also want to take into account the number of students that are not in both courses : $diff$. For example use the ratio:

$$measure_5 = \frac{diff}{inter + \epsilon} \tag{3.9}$$

But this measure would not be very representative of the similarity of two courses. Let us imagine that we have two courses with the 50 same students. They have a distance of 0, which seems fair. But now consider a first case where there is one student not in the intersection and a second case where there is two such students. between the two cases the score will be multiplied by 2 although intuitively the similarity has not changed a lot.

We have thus a collection of potential different measures. When optimizing the method we will evaluate the performance of those measures to see which one is the most appropriate.

Linkage type

The type of linkage is also an important parameter in hierarchical clustering. The linkage gives a precise meaning to the ambiguous notion of "distance between two clusters". There are 3 types of linkage : "single", "average" and "complete". The single linkage considers the distance between two clusters to be the shortest distance between pairs of points across the two clusters. The average linkage considers the average distance between all pairs of points. The complete linkage takes the maximum distance among all pairs. The danger of single linkage is that it could induce "long thin" clusters where close courses are like links in a long chain. The danger of complete linkage is its sensibility to outliers. Anyway the linkage is a parameter of the clustering and will be tuned later.

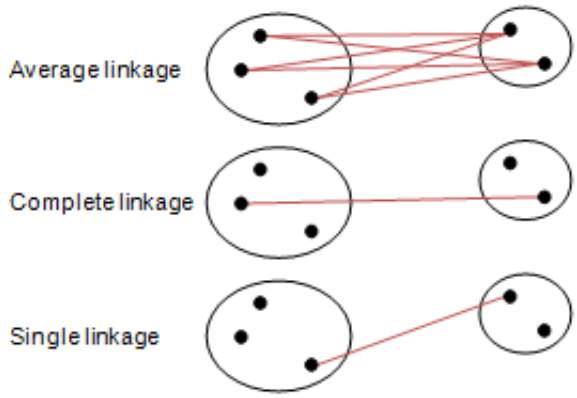


Figure 9: hierarchical clustering : linkage

Cutting level

Once the hierarchical algorithm has been performed, we get a tree of clusters that stores the history of agglomeration. We have now to decide at what point of the agglomerating process we stop. Going back to Figure 8, cutting the tree is drawing an horizontal line in the dendrogram. The clusters associated with the cut are represented by the lines segments that are intersected by the cut. To know which samples are in each cluster, one just follows the line segments of that cluster downwards along all ramifications until the x-axis.

Dendrogram of the data-set

On figure 10, you can see the dendrogram corresponding to the data-set of the Januar exam session. We can see that the distance measure (from equation 3.7) seems relevant. Indeed among other, we can see that the courses of the first grade are so close that the level joining them is very low. We can also see that the courses of the computer science option are together, that the courses of the engineers in computer are together and that they join a little bit higher to form a common cluster. This reflects very closely the reality. The courses of the "computer sciences" group are :

- computer law contract
- Object Oriented Programming Project
- Object Oriented Software Engineering
- management for computer scientist

The courses of the "Engineer in computer sciences" group are :

- Network
- Discrete optimization
- Calculability

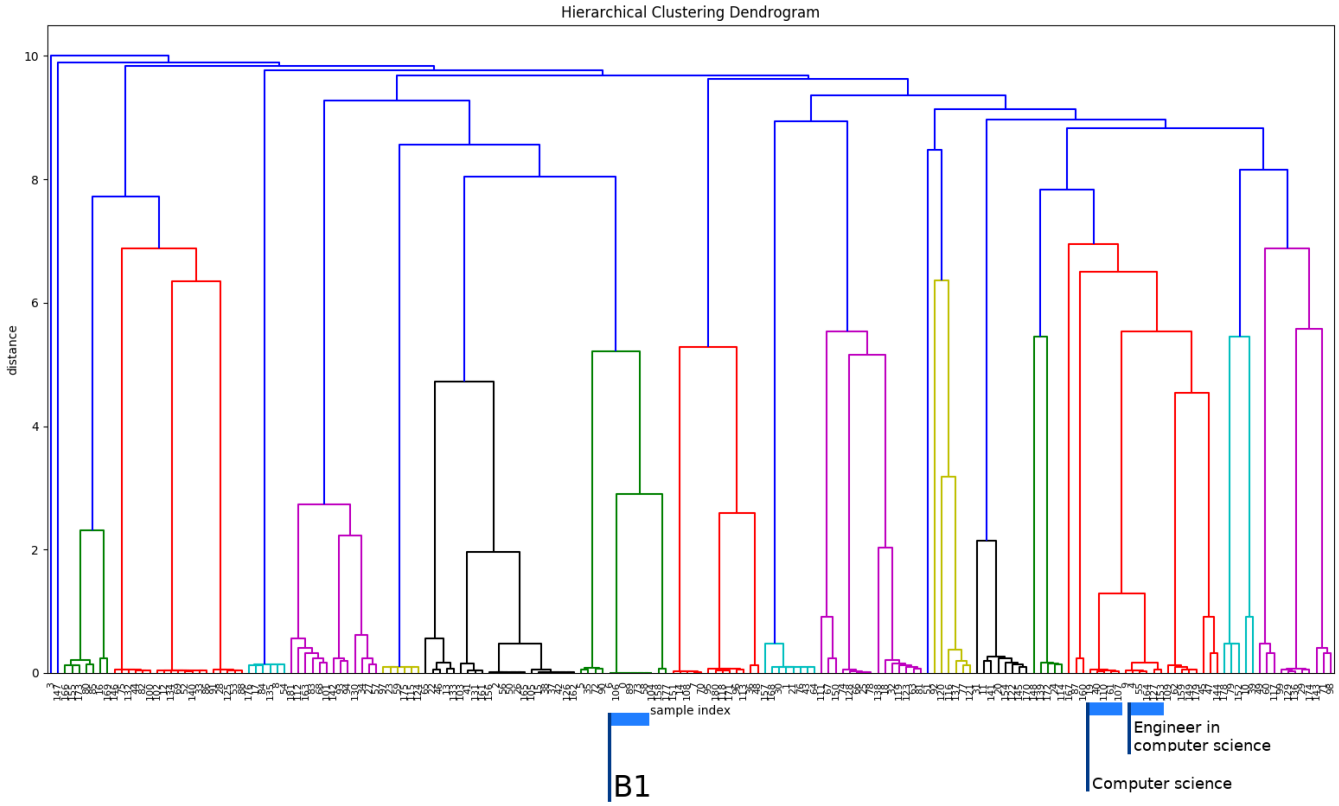


Figure 10: hierarchical clustering on the full data-set

- Logic
- Computer vision
- Machine learning

3.3 Iterative process : general idea

We combine two techniques to solve our problem : "Clustering" for choosing a group of coherent courses and then "MILP solving" for partitioning this group (partitioning a group means partitioning the oral courses of that group). The clustering will select a sub-problem and the MILP model will make the partitioning of this sub-problem. So in order to partition the whole data we must repeat this process on different subsets of the problem until all the courses that need partitioning have been partitioned. Figure 11 is a diagram showing an overview of the way both tools work together. The rectangle are operations, the diamond is a decision, the oval is a variable.

Initialization The first step is to initialize the *Partial Solution*. At the beginning no course is partitioned.

Iteration

1. We look in the partial solution if there are oral-courses that have not yet been partitioned. If all oral-courses have been partitioned we have finished.

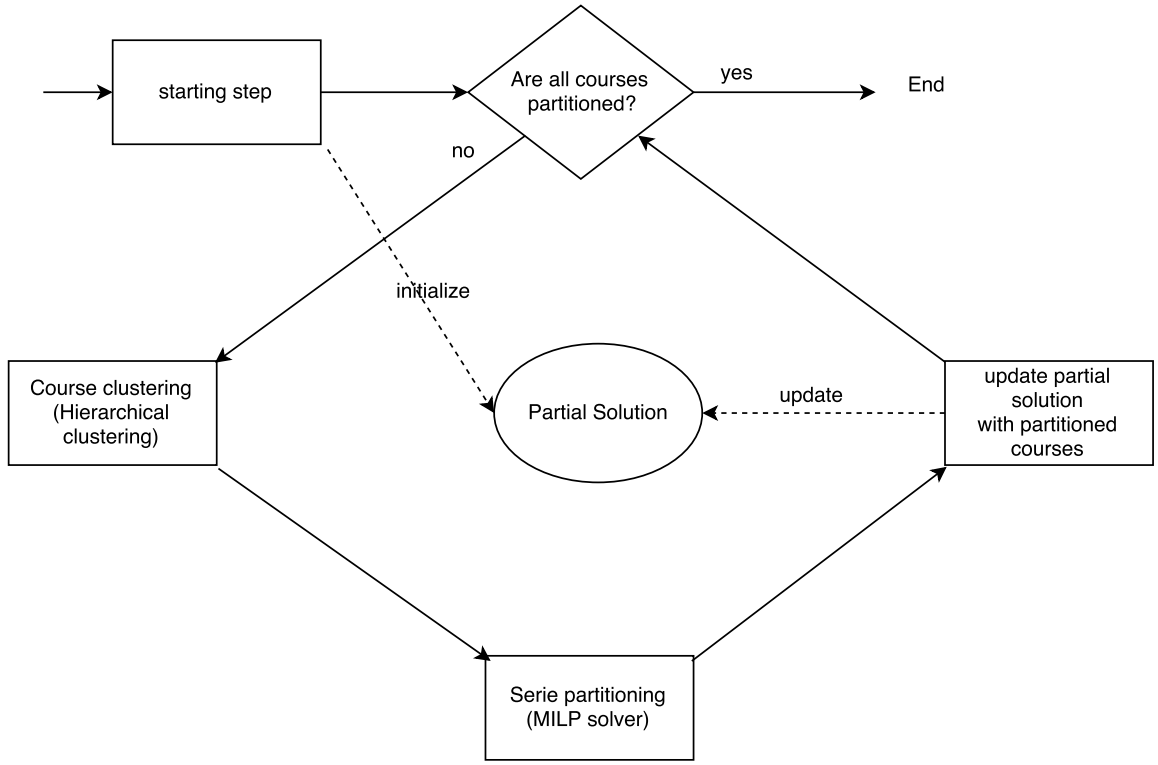


Figure 11: simplified method diagram

2. From the partial solution we do the clustering and select the "best" cluster

3. We perform the partitioning of the courses from the "best" cluster using MILP solver.

4. We update the partial solution

Ending We perform validity checks on the final solution to ensure that all constraints are satisfied.

As we can see on the diagram the course clustering operation is done at each

3.4 Sub-group selection : First approach

How to use hierarchical clustering to find the "best" group of courses to be partitioned? The first part of the problem consists in finding the best parameters of the clustering : the distance measure and linkage type. Intuitively, we would choose the distance measure defined in equation 3.7 because it combines both the proportion and the number of common students. This parameter (choice of the measure) however has to be correlated with the score to find out the best choice. It is also the case for the choice of the distance measure. We will cover that in section 3.8.

Then we must select the best cluster among all possible cutting level. As we said earlier, this choice is a compromise that must be found between large cluster leading to better result and taking more time and small cluster leading to faster but less accurate results. The challenge is thus roughly to find the biggest easily solvable cluster.

As a first approach let us cut the tree in a way that all clusters have a maximum (mean-)intra-distance i.e. impose a bound on the sum of the distances between all pairs of courses inside each cluster. This should intuitively impose all clusters to be small enough to be solvable. The value of the bound is a parameter of the method to be tuned. When we have identified the highest cut that guarantees the bound on each cluster, we then select one cluster that contains oral courses that have not yet been partitioned.

This approach works to some extend. For not too large data-set, it finds optimal or close to optimal solution in little time compared to the exact model. But it presents also some problems. We observe that in practice the intra-cluster distance is not always a good indicator of the complexity of the partitioning problem. Some small clusters can have however common students with a lot of other courses (most of them outside of the cluster). So even if the cluster is small the partitioning problem can be difficult. Those specific partitioning can take several hours. It highlights the downside of MILP model : when the problem is not very linearly relaxable its complexity is exponential with respect to the size of the data. Therefore from a certain size, the problem become unsolvable. We have thus to find another approach to avoid this exponential explosion but keeping a good quality of solution.

Before presenting the other approach, it is a good idea to speak about the cluster partitioning and more specifically about the way we build the data that we will give to the MILP model.

3.5 Sub-group solving (using Mathematical Integer Linear Programming)

For solving the partitioning of a cluster we use the model that was defined in section 2.5. Thus we need to create the data of the model from our partial solution. Here is a recall of those data.:

$$courses\_student_{c,s} = \begin{cases} 1, & \text{if student } s \text{ follows course } c \\ 0, & \text{otherwise} \end{cases} \tag{3.10}$$

$$courses\_exam_{c,x} = \begin{cases} 1, & \text{if exam } x \text{ is associated with course } c \\ 0, & \text{otherwise} \end{cases} \tag{3.11}$$

$$capacity_x = \begin{cases} max > 0 (\in N), & \text{if exam } x \text{ has a maximal capacity of } max \\ -1, & \text{if exam } x \text{ is not bounded} \end{cases} \tag{3.12}$$

$$exam\_pairs_p = \{x, y\}, \quad x \text{ and } y \text{ being the index of the exams of pair } p \tag{3.13}$$

The "Pairs" are all pairs of exams that could *potentially* share common students and whose compatibility depends on the partitioning that will be done at this iteration. As a reminder, a pair of exam is *compatible* if the two exams share no student and *incompatible* otherwise. Let us explain this more in detail.

In the partial solution we have three kind of courses:

- written courses
- oral courses that have not been partitioned yet
- oral courses that have already been partitioned

For the first and the last category, the exam(s) corresponding to those courses are known. Their composition will not change in the present or future iterations. For the second category, we don't know yet how the students will be split into exams.

When it comes to computing the pairs, we must only take into account the pairs of exams such that at least one exam of the pair will be partitioned during the solving of this iteration i.e. at least one exam of the pair is associated to an oral exam that is in the cluster and that must be decomposed into more than one series. All

the other pairs are of no interest as their composition will stay the same during the cluster partitioning and thus will not influence the objective of the model.

We observed that the execution time of the model is closely related to the number of pairs meaning that the number of pairs is a good indicator of the time that the model will need to be solved.

3.6 Sub-group selection : second approach

Now that we have clarified the notion of pairs, let us present the second approach which is more pragmatic and efficient than the first.

The downside of the first approach is that the intra-distance of the cluster to be partitioned is not a good indicator of the complexity of the MILP partitioning model. This time we will use the number of pairs of the MILP formulation as a criterium for choosing the cluster to be partitioned. This avoids any exponential explosion of the MILP sovling.

Previously we defined a fixed upper bound on the intra-distance of the clusters. This bound determined the "best" cutting level. In our new approach, as we can evaluate more precisely the complexity of a given cluster partitioning (using the number of pairs of the corresponding MILP model) it would be too restrictive to fix the cutting level as we did in our first approach. Instead we will find the highest cutting level so that at least one of its clusters can be solved efficiently. Choosing the appropriate cutting level becomes more complex than in the first approach. If we wanted to do it exactly, we should test every cluster from every cutting level. For each possible cluster compute the MILP formulation to get the number of pairs (as a complexity measure) associated to that cluster. Then among the cluster that have a "low enough" complexity, pick the one from the the highest cutting level. Doing so we could choose the best possible cluster to be partitioned but the process would be extremely long. We must find a way to speed up the research of an "optimal" cluster.

You can refer to section 3.7 (and Figure 12) to see an overview of this method.

Firstly, for each cutting level, we will only consider the cluster having the smallest intra-distance so that we have to test only one cluster per cutting-level. Taking the smallest cluster seems relevant : the average cutting level of the selected cluster is higher and corresponds to a more general problem (this is approved by experimental tests). We can define the score of the cluster of a cutting level as a function of this cutting level. A higher cutting level is likely to lead to a bigger score. For finding the best cutting level (the highest cutting-level which is solvable by the MILP), we proceed by dichotomous search, by trial and error reducing the searching range at each step. Let us say levels goes from 1 to n, we define a lower and upper bound for the dichotomous search which we initialize respectively to 1 and n. Iteratively, we test a level in the interval between the two bounds. Depending on the result of this test the lower or the upper bound takes the place of that intermediate level. We repeat the process until our two bounds meet. Consequently, by testing only $log(n)$ levels we can find (more or less) the best one. It is not exact for the reason that cluster-complexity is not exactly a monotonously increasing function of the cutting level. But in practice it gives good results.

But a problem can still occur : in some cases even at the first level (one cluster per course) the clusters are unsolvable for the same reason as in the first approach (section 3.4): there are a too many pairs between the exams of the course of the cluster and the exams corresponding to courses outside the cluster. In this situation, what we do is ignoring some outside-clusters courses so that the complexity is reduced to some acceptable level. More precisely we ignore progressively the courses having the highest distance with the course of the cluster until the complexity (number of pair of the MILP model) comes below the accepted limit. To do this we use again dichotomous search to reduce logarithmically the complexity. In this case we know that the MILP complexity with respect to the number of outside-courses that are considered is a monotonous function. Thus we will find the best value.

We can now solve the partitioning problem for the cluster that we have selected.

3.7 Iterative process : implementation

The process that we have described selects the most appropriate cluster and partitions its oral courses into series. In order to get to the final solution, we must repeat this process several times. At the very beginning of the method, we start by defining a partial solution where no course is partitioned. Then after each iteration this partial solution is updated. When all courses have been partitioned it means that the problem is finished. When, at the end of an iteration, a course is partitioned into several series, each series will in later iterations be considered as an exam who do not need to be partitioned similarly as the written exams.

We can see on Figure 12 a more detailed diagram of the method. The rectangles represent operations, the ovals represent data. The dashed arrows show the relations between the operations and data: the dashed arrows entering operations mean "input", the dashed arrows leaving operations mean "output". The continuous arrows show the order in which the operations are performed. When an operation has several possible next operations, each outgoing arrow is labeled with the corresponding choice.

3.8 Method parameters tuning

The parameters of the method are the following:

- distance measure for clustering
- type of linkage for clustering
- maximum number of pairs allowed in MILP model.

Those parameters must be tuned in a way that optimizes the method. The best parameters are those who lead to the best score but at the same time who do it in a reasonable amount of time. This problem is actually difficult because the parameters can have dependencies. Moreover we optimize the parameter with respect to the data-set that we have. But this data-set is not necessarily representative of an "average" data-set. Therefore it is useful to introduce some theory to know what we are doing and to avoid having a wrong methodology.

Theoretical consideration

Our method depends on several parameters. The goal of parameter tuning is to find the set of them that gives the best score on an "average" exam session. Ideally we would need an infinite number of data-sets to be able to tune our method independently of a given data-set (a data-set being the curriculum of all students for a given semester). For each data-set we should apply our method for each combination of the parameters and express the score as a function of those parameters. Then we should average this function on all data-set to avoid over-fitting our parameters to a specific data-set. But this is in theory. In practice we have a limited data-set. To simulate a bigger set of data we can use a sort of bagging (boostrap aggregating) technique. But first let us clarify the notions of sample, data-set and score in the context of this problem.

Let us use the terminology of machine learning. For a given set of student-courses of one semester, there is one optimal score (corresponding to one or maybe several serie partitioning). This means that a set of student-courses is to be considered as one "object" and the "output" corresponding to that object is the optimal score. In an ideal world we should have an infinity of objects (i.e. a infinity of sets course-student) and for each object we should know the optimal score. In that case, we could test our method on each object and tune its parameters so that the score of our method is in average as close as possible to the known optimal score of each object.

To solve the problem of over-fitting (we have only 2 objects (January and June) instead of an infinity) we could create artificially new objects by picking random sub-sets of courses from one of our 2 objects. This would give us several new objects. But they are not really independent and moreover if we pick too little courses from a data-set, we could end up with objects that are not realistic. Let us illustrate that with an extreme example: if we pick only two courses from a data-set, it is likely that those two courses will correspond to different curriculum and in that case the problem becomes trivial because there is no common students. So this kind of bagging technique has limitations that we must be aware of.

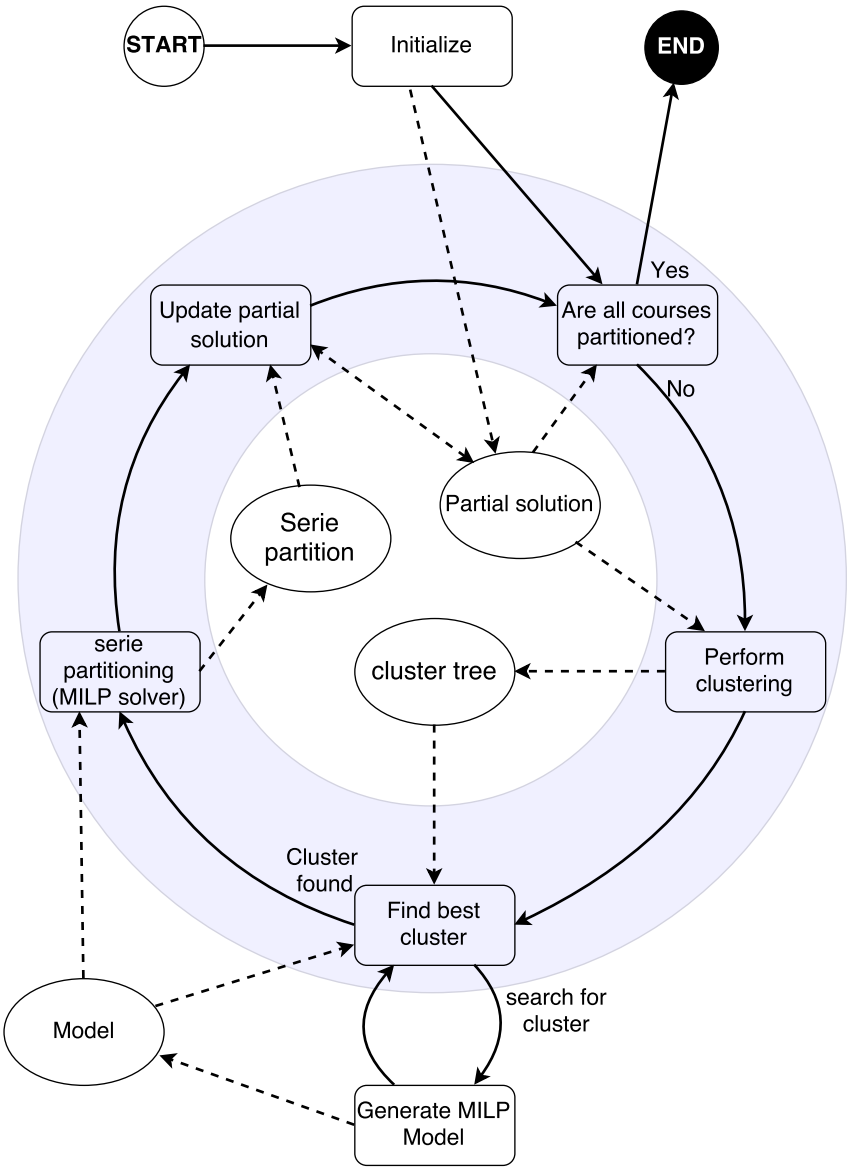


Figure 12: detailed method diagram

Now that we know the risk of over-fitting let us come back to parameter tuning. In order to find the best parameters we should ideally create a function $score(p_1, p_2, p_3, ...)$ where p_i are the different parameters of the method. Then we have to pick the minimum of that function. But in practice such a function is impossible to compute because there are too many combinations of parameter values and, knowing that running the method takes times ($\approx$ 20 minutes), it is impossible to run it around 1000 times to test every parameter combination.

To be able to tune our parameters in a reasonable time we could make the assumption that the parameters are independent. This is most likely false but if they are not too strongly dependent, this could in practice allow us to gain some accuracy. If we assume the independence of the parameters, the score function becomes :

$$score(p_1, p_2, p_3, ...) = f_1(p_1) \,.\, f_2(p_2) \,.\, f_3(p_3) \,... \tag{3.14}$$

(We will use lowercase for variable and uppercase for values)

So if we find $p_i = P_i^{optimal}$ that minimize the score function (for a set of value of P_k $\forall k \neq i$ such that $f_k(P_k) \neq 0$) then we know that $P_i^{optimal}$ is the optimal value in general.

In other words we just need to optimize each parameter independently to find the global optimum.

Tuning distance measure

Let us evaluate the effects of the distance measure of the clustering technique on the final score. In order to find the best measure, it is convenient to have a parametric measure. Doing so we can express the score as a function of the measure's parameters and look for the minimum of this score-function within a region of the parameter's space. The parametric measure we will use is the following:

$$measure_2 = \left(\frac{inter^a + \epsilon}{union^b} \right)^{-1} \tag{3.15}$$

We will set the value of ϵ to 0.0001 and tune the measure with respect to a and b. a and b are likely to be dependent. a will take values from $\{0.125, 0.25, 0.5, 1, 2, 4\}$ and b from $\{0.5, 1, 2, 4, 6\}$. which makes 30 combination of values. The value of the other parameter (fixed) are:

- max_pair = 65
- linkage = average

Each combination of parameters is executed 4 times on randomly created objects to reduce over-fitting. Each object draws randomly 98 courses out of 140 (principle of bagging). It is the same set of 4 objects that is applied on the 30 sets of parameters. In total we have 30*4 = 120 executions of the method.

score:	a\b	0.5	1	2	4	6
	0.125	402.5	401.75	401.75	403.25	403.25
	0.25	402	402.5	401.75	401.75	403.25
	0.5	402	402	402.5	401.75	401.75
	1	404	402	401.75	402.5	402.75
	2	416.25	415	408.25	401.75	401
	4	421.25	422.25	423.75	417.5	412

We see that there are a region of the parameter to be avoided (high a value - low b value) but otherwise there is no clear minimum. Moreover there seems to be a correlation between the a and b : the score seems to depend on the ratio $\frac{a}{b}$ at least outside the area where the score becomes bad. Indeed if we go along diagonals we have little variations in the score.

Here are the times of execution:

a\b	0.5	1	2	4	6
0.125	15.2	11.6	11.6	12.3	12.2
0.25	15.1	12.8	12.0	12.2	11.8
0.5	11.6	15.1	13.0	12.1	12.1
1	16.3	11.4	15.7	12.8	12.1
2	7.7	7.6	7.8	10.3	10.2
4	9.5	10.2	7.1	10.3	7.7

Let us do again the parameter tuning but differently. As the ratio $r = \frac{b}{a}$ seems to have some meaning let parametrize the measure distance this way:

$$measure_2 = \left(\frac{(inter+\epsilon)^a}{union^{a.r}}\right)^{-1} = \left(\frac{(inter+\epsilon)}{union^r}\right)^{-a} \tag{3.16}$$

Score:

a \r	2	3	4	6	8
0.25	365.75	365.25	366.5	366.5	365.5
0.5	365.75	365.25	366.5	365.25	365.5
1	365.75	366.5	366.5	366.5	366.25
2	367.25	367.5	367	366.75	366

Execution time:

a \r	2	3	4	6	8
0.25	37.4	28.7	33.2	35.1	31.0
0.5	31.9	33.1	35.0	34.5	29.4
1	31.0	31.3	31.2	33.4	34.2
2	35.7	39.2	37.3	37.8	37.8

We conclude that the best parameter (a,r) are (0.25,3). Thus the best parameter (a,b) are (0.25,0.75).

Let us tune the value of ϵ. We observe that smaller values gave better results but that from 0.1 it made no difference:

ϵ	0.005	0.01	0.05	0.1	0.5	1
score	1802	1802	1802	1802	1803	1804

Tuning the maximum number of pairs

This parameter is different than the others in the sense that increasing it can only enhance the score (unless we have already reached the optimum). So tuning it is not a matter of looking for the best score. The goal is to find the best compromise between efficiency and speed.

parameter range	observation
50 - 180	The score decreases well. In the mean time the execution time increase but stay in a reasonable range (a few minutes)
180-230	the score continues decreasing a little. The time increases more (up to 30 minutes).
>230	It takes really too much time and for this reason, it becomes difficult to make benchmark
∞	Corresponds the exact model with no clustering

3.9 Conclusion of the Method

As a conclusion we can say that the method, although a bit complex, works very well. It runs in reasonable time (around 1700 sec. for the whole problem) an achieves seemingly good results. For problem-sizes that are

at the limit of what we can solve using the exact MILP model, the clustering-MILP method achieves the same optimal score but in much shorter time which is very promising.

4 Meta-heuristic : simulated annealing

The Simulated annealing method belongs to the family of meta-heuristic methods. Those generic methods aim at finding the global extremum of some cost function defined on some space using iterative stochastic algorithms. They are mostly used for highly complex problems for which there is no efficient classical approach.

4.1 Principle of the method

The simulated annealing principle is the following: it starts with a valid state (it can be chosen at random). At each iteration it will randomly choose a valid neighbouring state and will favor the ones decreasing the objective, making the algorithm (hopefully) converge toward the global optimum; "valid state" means satisfying the constraints of the problem. The susceptibility of the algorithm of accepting a state with a higher objective is high at the beginning of the algorithm. It makes it possible to explore the space of states without being stuck in some local minimum. As the computation goes this susceptibility is progressively reduced. To express this susceptibility, we use the notion of temperature (by analogy with the annealing process in metallurgy). We begin with a high temperature and decrease it at each iteration. A zero temperature means that the probability to accept a "higher state" is equal to zero.

This method fits more or less any optimization problem and is easy to implement. The difficulty resides in the parameters of the method that must be calibrated carefully.

4.2 Method parameters

One parameter of this method is the temperature law as a function of the iteration. The other parameter is the stop criterium. Common temperature laws are $T(k) = T_{init}/k^i$ or $T(k) = T_{init} \, . \, r^k$ where k is the iteration number and $i > 0$ and $r(< 0)$ are parameters. We could simply stop the computation by imposing $k \leq K_{max}$. What I did was allow a maximum number of iterations between two improvement of the objective. So when the objective decreases too slowly the computation stops.

4.3 Implementation of the method to serie partitioning problem

The whole point of this method is to be able to test a lot of configurations per unit of time. It is therefore important to optimize the computation complexity of each iteration. A basic approach would be to look for a valid neighbour state, compute its score and then decide to accept it or not according to the temperature. This induces the computation of a whole state in memory and the computation of its score. Those are heavy operations. In my implementation, instead of computing the whole neighbour state, I only compute a difference and compute the new score based on the previous score and a state difference. I compute the new state only if it is accepted, which happens very few as soon as the temperature gets low. This makes the whole process very fast.

4.4 Conclusion of the method

This method has 2 main advantages : it is very easy to implement and can be parallelized as much as we have cores. But the disavantages is the fact that it does not necessarily converge to a global optimum. Compared to the Milp-cluster approach it takes a lot more time to get to a result that is a bit worse. But if we let it run long enough the difference with the first method is generally small or equal to zero. In all the runs that I performed it was never better that the Milp-cluster approach in term of score.

5 Simplified Schedule model

We have now two methods partitioning students into series. The objective of this thesis seems thus fulfilled. Yet we have not entirely finished the job. Indeed those methods are only one half of the larger problem of making schedules. We can not assure the quality of the methods if they are not tested with a schedule maker. Even if the score that was defined seems to make sense it is however an abstract quantity and does not constitute the proof of the quality of the partitioning. Therefore we have to build a schedule maker and interface it with our series-partitioning methods. Doing so we can see how the score for the series affects the quality of the computed schedule and get a tangible result testifying the pertinence of our methods.

We do not however build a sophisticated schedule maker taking all real-case subtleties into account. It would be too much work and it is not the purpose of this thesis. Moreover we do not need such a program to validate our methods. We can satisfy ourselves with a simple model but yet giving us some meaningful values. Our model will define a series of time-slots and will assign all exams to those slots respecting the constraint that two conflicting exams can not be in the same slot while trying to minimize the total number of used slots.

5.1 Model definition

Data

list of pairs (i,j) : for all exams i and j that can not be in the same slot.

Variable

$assign_{i,k} = 1$ if exam i is in slot k, 0 otherwise.
$used\_slot_k = 1$ if slot k is used by at least one exam. 0 otherwise.

Constraints

An exam occurs only once:

$$\sum_k assign_{i,k} = 1 \quad \forall\, i \tag{5.1}$$

Two incompatible courses may not occur at the same slot:

$$assign_{i,k} + assign_{j,k} \leq 1 \tag{5.2}$$

$\forall k$ and $\forall$ pair (i,j) of incompatible exams

Defining $used\_slot_k$ variable:

$$used\_slot_k \geq assign_{i,k} \quad \forall k, i \tag{5.3}$$

objective

$$\min \left\{ \sum_k used\_slot_k \right\} \tag{5.4}$$

5.2 Differences with the realistic model

constraints

In the realistic model we have to take into account the availability of the teachers. We must impose that some exam must take place after some other. We should take into account the number of available exam rooms for big groups of students.

objective

The objective is more complex to encourage schedules that gives the students space between their exams.

6 Results

6.1 Comparison of solving methods

Let us now compare the scores obtained with ours different methods. On Figure 13, you can see the scores obtained for 4 different series partitioning techniques:

1. Clustering + MILP (blue)
2. Simulated annealing (orange)
3. Partitioning student without any optimization (using the order of the file) (yellow)
4. Random partitioning (green)

on 4 different size of problem :

1. 30 courses
2. 60 courses
3. 100 courses
4. 182 courses (the whole session)

For the first three sizes of the problem, which are actually sub-problems, the method was used on several different random groups of courses to reduce the variance relative to a specific group. The problem of size 30 was tested over 6 different groups, size 60 over 4, and size 100 over 2.

The 5th column in the histogram (brown) represents the difference between the simulated annealing and the cluster+milp method, magnified by 100.

6.2 Effects of series partitioning score on exam scheduling

In the introduction, we raised the question of the validity of the score defined for the series partitioning. Let us answer this question using our statistical data. To prove that the choice for our score was relevant, we must show that the quality of the score for the partitioning affects the score for the schedule creation. In other words, we must observe a clear correlation between the two scores.

Let us compute the statistical correlation between the score of the serie partitioning and the score of the simplified schedule model. We generate N various series partitioning of our data-set. $X_i \quad (i \in [1, N])$ is the score of the i^{th} partitioning. We make such that the scores X_i are as diverse as possible ranging from the near the optimal to the worse random solution. For each serie partitioning i we can compute the corresponding optimal simplified model and record its score in the variable Y_i.

We will use the Pearson correlation coefficient as a correlation measure:

$$\rho(X, Y) = \frac{COV(X, Y)}{\sigma_X \,.\, \sigma_Y} \tag{6.1}$$

The covariance $COV(X, Y)$ is the expectation of the product of the two variables:

$$COV(X, Y) = E\big\{(X - \mu_X) \,.\, (Y - \mu_Y)\big\} \tag{6.2}$$

where:

$$\mu_X = E\big\{X\big\} \text{ and } \mu_Y = E\big\{Y\big\} \tag{6.3}$$

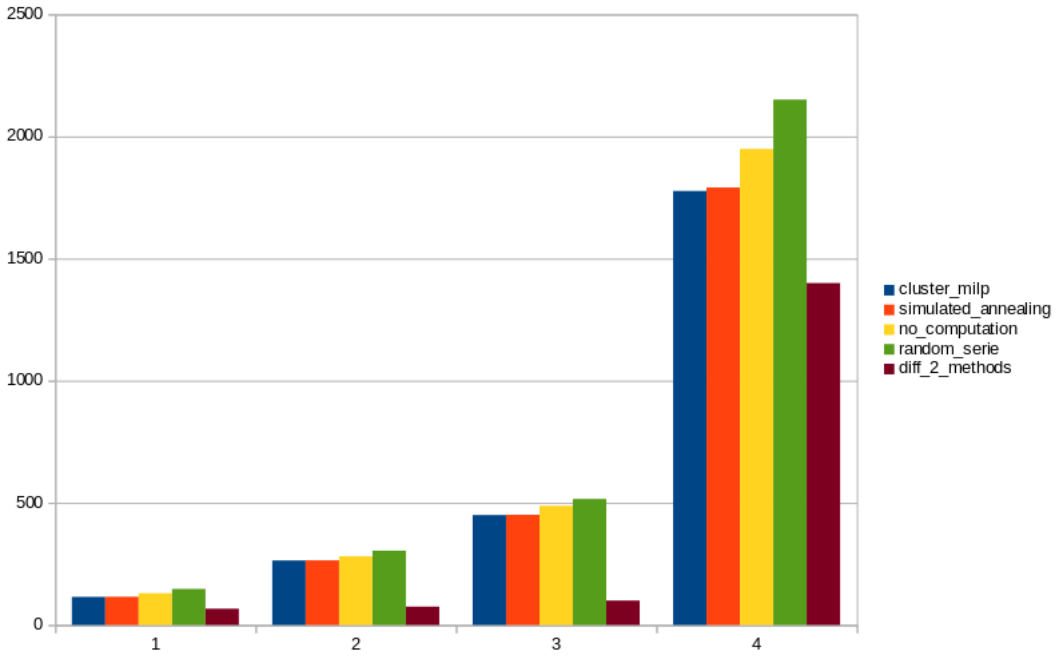


Figure 13: Methods comparison : Score. X-axis: samples, y-axis: score. The brown value shows the difference between the orange and the blue, magnified by a factor 100.

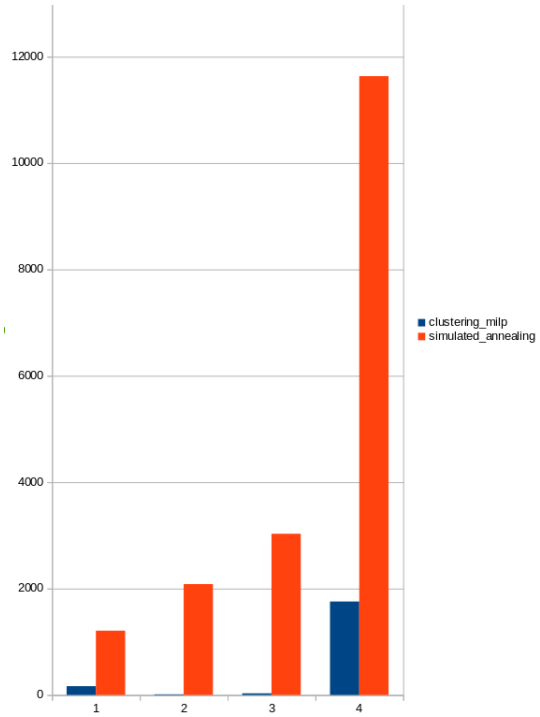


Figure 14: Methods comparison : Execution time. X-axis: samples, y-axis: time in seconds. Blue: clustering+MILP method, Orange: simulated annealing method

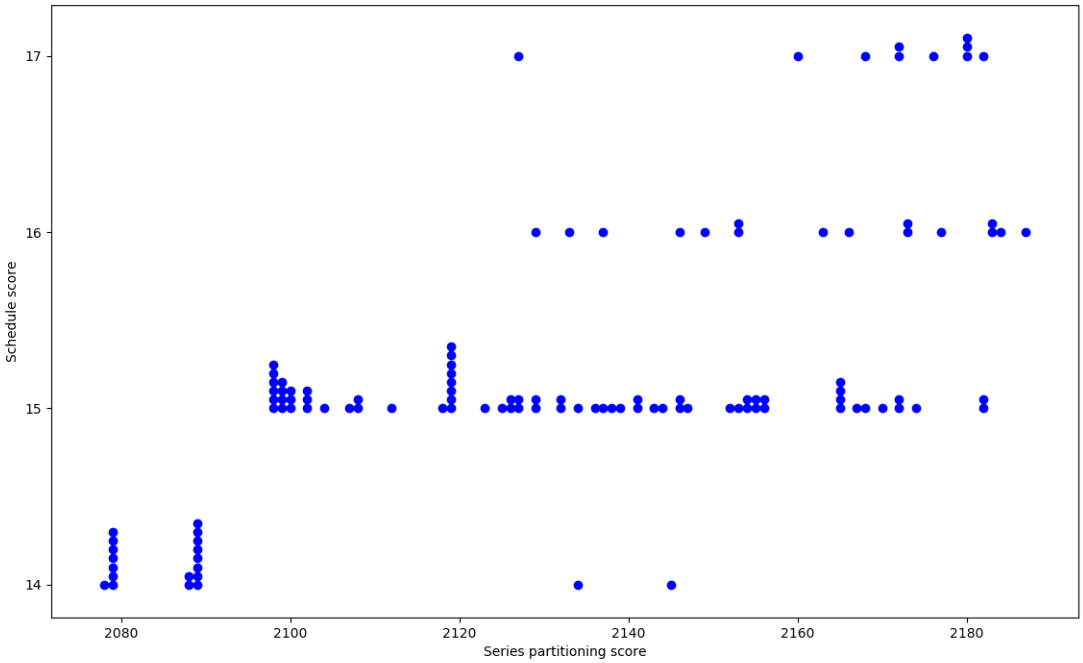


Figure 15: Correlation between scores (superposed points have been piled for lisibility)

The standard deviation σ_X is the square root of the variance of X

$$\sigma_X = \sqrt{E\{(X-\mu_X)^2\}} \text{ and } \sigma_Y = \sqrt{E\{(Y-\mu_Y)^2\}} \tag{6.4}$$

If we apply those formula on our data we get:

$$COV(X,Y) = 12.87 \tag{6.5}$$

$$\sigma_X^2 = 793.02 \tag{6.6}$$

and

$$\sigma_Y^2 = 0.626 \tag{6.7}$$

$$\rho(X,Y) = \frac{12.87}{\sqrt{793.02\ 0.626}} = 0.577\ >\ 0 \tag{6.8}$$

which is positive and clearly different that zero. We can thus conclude that our choice for the score of the series partitioning was good.

7 Conclusion

I have started this project with an open statement : find a way to solve the series partitioning problem. The first difficulty was to define exactly the problem. Indeed, being a sub-problem to a bigger one, its definition and formalization needed first to have a good knowledge and understanding of the whole problem. After some preliminary work and considerations on the whole problem, I could give a formal definition of our problem and define mathematically a score (depending on the partitioning) to be minimized.

The first approach was to express the problem as a MILP model and to solve it as such. This showed us the great complexity of the problem and that it needed different approaches. But it also gave us some exact solutions for smaller data-sets that we could use as benchmarks for later methods.

I firstly explored an hybrid solution combining hierarchical clustering and MILP solver. Properly tuned it achieved very good results. But it was difficult to evaluate its precision because of the absence of references for the whole problem. For this reason and also for the sake of scientific interest we explored a second method using a contrasting approach.

The simulated annealing, which is part of the meta-heuristic methods, worked well and accordingly to what we could expect from it: it was simple to implement, required little memory usage, but needed a lot of time to get to a result equivalent to what we had with the other method. Meanwhile it was a very convenient tool for generating a lot of feasible solution of different qualities.

The two methods a very consistent regarding the results they produce. Although the first method gives better results the difference is always small. This gives some confidence about the validity of both methods. Indeed if one or two of the methods were diverging from the optimal solution it would be very unlikely that they produce correlated results.

I achieved solving the first sub-problem of the schedule making, the series partitioning problem, by defining a score to reduce and by finding efficient methods minimizing that score. But most importantly I had to show that the way I defined the problem was coherent with respect to the subsequent problem. To prove this I build a simplified solver of the subsequent problem, the schedule maker. I could verify that indeed the quality of the score for the first problem enhanced the quality of the final schedule.

8 Hardware and technologies

The implementation of this problem is done with the Julia programming environment and libraries. Interfacing with Cplex solver is done through Julia JumP library. Execution of the codes has been done on *Euclide* computer of the faculty. The processor (intel i7) has 12 cores turning at 3.47GHz. It has 24.5 GB of RAM memory.

Julia is a programming language that I discovered doing this thesis. I enjoyed a lot using it. It is in my opinion the best scientific-oriented programming language that I have experienced during my studies. It combines the syntactic/semantic rules and basic library-functions specific to scientific computing (very close to matlab) with features inspired from recent and serious languages making it very fast and general purpose. The downside of its recentness is that finding help on the internet is not as easy as with more common languages like java or C although there is a good documentation page.

9 Acknowledgement

I would like to thank my mentor Quentin Louveaux for his availability during the whole time of my thesis, for all the discussions we had that helped me to lead my work in the right direction. I would also like to thank all

the people who contributed to the open-source/free technologies that I used daily for this thesis : Linux-GNU system, Julia, Python, Atom editor, Firefox and Sharelatex, draw.io diagram editor.