School of Business, Economics and Information Systems

Chair of Management Science/Operations

and Supply Chain Management

Research Paper

Practical Course: Management Science

Case Study - Student sectioning

Chairholder: Prof. Dr. Alena Otto

Supervising tutor: Ayse Karabayir

Edited by:	[redacted]	Vincent Haunberger
Matriculation number:	[redacted]	[redacted]
Course of studies:	B. Sc. Information Systems	B. Sc. Business Administration and Economics
Semester:	3	6
E-mail:	[redacted]	vhaunberger@gmail.com

Passau, 11.08.2022

Outline

1 Problem overview

Student sectioning problems describe the assignment of students to their courses of preference given certain boundary conditions such as the maximum number of participants per course. Solving such problems is not trivial since there are many factors which can influence the allocation.

In the following work a generic integer optimization problem is solved where a set of students ῝Ο should be optimally allocated to their preferred courses while respecting the capacity of the courses ά ῶ ώ ö ò ò Ὠ. Firstly, mathematical models that build the base of the optimization problem are set up. Following, these models are implemented into IBM ILOG CPLEX Optimization Studio to calculate optimal solutions. Finally, these solutions and our models are then analyzed and discussed.

2 Mathematical model

In the problem, students ῞Ωᴺ ῞Ο choose to attend a certain number of courses ὲ ῷ ῦ ὰὲ ῶ Ὼ ὅ έ ῤ that they can select from a set of courses ὐ with each individual course ῏Ωᴺ ὐ. Students can review the courses beforehand and rank them based on their personal preferences (ὴ ὶ Ὠ῞Ω Ὠ ἴ Ὠ). Preferences for the courses are strictly ordinally scaled, which means they must not select the same value for the same course. The highest rank has the value 1, the second highest rank has the value 2, and the lowest rank the value ᾠ. Furthermore, students can state how many courses they want to visit (ὲ ῷ ῦ ὰ ῶ ὲ ὲ Ὠ Ὡ). The maximum number of attendees for a course ῏Ω is given by ά ῶ ώ ö ò ò Ὠ. The decision variable ώ indicates whether a student visits a course or not and ώ states if a course is being overbooked or not.

The following decision variables have been used:

$$
x_{ij} = \begin{cases} 1, & \text{if student attends the course} \\ 0, & \text{else} \end{cases}
$$

$$
y_j = \begin{cases} 1, & \text{if course is being overbooked} \\ 0, & \text{else} \end{cases}
$$

The following table provides an overview over all used variables:

Variable	Description
$I = \{1, ..., n\}$	Set of all students i
$J = \{1, ..., m\}$	Set of all courses j
z_j	If course j takes place or not
$nbPlannedCourses_i$	Number of courses a student i wants to attend
$preferences_{ij}$	Student i's preference ranks of all courses $j \in J$
M	Penalty cost for not assigning a student to the number of planned courses
$nbExtraSlots_j$	Number of additional slots per course j
$penaltyExtraSlots_j$	Cost for booking at least one extra slot of course j
$minAttendees_j$	Minimum number of participants for a course j
$maxAttendees_j$	Maximum number of participants for a course j (Course capacity)

The main model is extended by certain options and additional constraints. Firstly, we provide the possibility to have student ῞Ω attend fewer courses than his preferred number of courses, ἐ ὦ ὖ ἀ ὤ ἐ ἐ Ὠ Ὢ. This deviation comes at a constant cost M, that is applied for each course less than ἐ ὦ ὖ ἀ ὤ ἐ ἐ Ὠ Ὢ. Secondly, we allow overbooking of a course, stated by ώ. Thirdly, we require a minimum number of students ά ῞Ω ἐ ὂ ὸ ὸ Ὢ to attend a course ῏Ω for it to take place. Lastly, the model is extended by allowing courses to be cancelled.

2.1 Main model

The goal is to maximize the satisfaction of students by ensuring that as many students as possible are assigned to the courses of preference. This is controlled by minimizing the product of assigned courses with the respective ἡ ὶ Ὠ῍Ὠ Ὠ ἲ Ὢ of ἐ ὦ Ὠ the course (1). Furthermore, each student must be assigned to the desired number of courses ἐ ὦ ὖ ἀ ὤ ἐ ἐ Ὠ Ὢ (2). Moreover, the maximum course capacity ά ὤ ὼ ὂ ὸ ὸ Ὢ of a course should be respected (3). The value range of the decision variable is set (4).

$$
\begin{array}{lllr}
\text{Minimize} & \sum_{i\in I}\sum_{j\in J} preferences_{ij} x_{ij} & & (1) \\
\text{Subject to} & \sum_{j\in J} x_{ij} = nbPlannedCourses_i & \forall i \in I & (2) \\
& \sum_{i\in I} x_{ij} \leq MaxAttendees_j & \forall j \in J & (3) \\
& x_{ij} \in \{0,1\} & \forall i \in I, j \in J & (4)
\end{array}
$$

2.2 Extension 1

The first modification of the main model is to allow students to attend fewer courses than their preference $nbPlannedCourses_i$, i.e. the boundary condition (2) will be replaced by (6). In the objective function (5) we introduce an additional term with the penalty cost M which is zero when all students i attend $nbPlannedCourses_i$ courses. If this is not the case the term is non-zero and therefore multiplied by M.

$$\sum_{i\in I}\sum_{j\in J} preferences_{ij}x_{ij} + \sum_{i\in I}(nbPlannedCourses_i - \sum_{j\in J} x_{ij})M \quad (5)$$

$$\sum_{j\in J} x_{ij} \leq nbPlannedCourses_i \quad \forall i \in I \quad (6)$$

2.3 Extension 2

In Extension 2, overbooking of courses is now possible. The maximum course capacity $maxAttendees_j$ is being increased by a number of extra slots ($nbExtraSlots_j$) (8). This is enabled by enlarging the course capacity be the number of additional slots. To keep overbooking at a minimum, a soft constraint is added. Each time a course has more students than expected the penalty ($penaltyExtraSlots_j$) must be paid once (7). The soft constraint weighs on the objective function, which is minimized. In the model the penalty does not depend on the number of extra slots booked. We also introduce a new variable which states whether additional slots are booked or not (y_j) with the range of values defined in (9). The more complex objective function is now given by (7).

$$\sum_{i\in I}\sum_{j\in J} preferences_{ij}x_{ij} + \sum_{i\in I}(nbPlannedCourses_i - \sum_{j\in J} x_{ij})M + \sum_{j\in J} y_j penaltyExtraSlots_j \quad (7)$$

$$\sum_{i\in I} x_{ij} \leq maxAttendees_j + y_j nbExtraSlots_j \quad \forall j \in J \quad (8)$$

$$x_{ij}, y_j \in \{0,1\} \quad \forall i \in I, j \in J \quad (9)$$

2.4 Extension 3

The model is now extended requiring a minimum number of students ($minAttendees_j$) to attend a course (10). This constraint prevents courses from taking place with an insufficient number of students. The objective function doesn't change in this step.

$$\sum_{i\in I} \geq minAttendees_j \quad \forall j \in J \quad (10)$$

2.5 Extension 4

According to extension 3, courses [illegible] may take place that are very unpopular and therefore increase the value of the objective function. To counter this behavior a new variable [illegible] is introduced which is 1 if the course takes place and 0 if the course is cancelled. In other words, we can now minimize the objective function by cancelling courses. Its range value is defined in (13). An unpopular course can now be cancelled and will not be respected in constraint (12). The objective function doesn [illegible] change in this step.

$$\sum_{i\in I} x_{ij} \leq (maxAttendees_j + y_j nbExtraSlots_j) z_j \quad \forall j \in J \qquad (11)$$

$$\sum_{i\in I)} \geq minAttendees_j z_j \quad \forall j \in J \qquad (12)$$

$$x_{ij}, y_j, z_j \in \{0,1\} \quad \forall i \in I, j \in J \qquad (13)$$

3 Results and analysis

The above-described mathematical model extended till extension 4 (see appendix Formulation 2) was implemented into the optimization software IBM ILOG CPLEX and used to calculate optimal solutions for different data sets. In the following section, these results as well as the corresponding parameters are analyzed. Additionally, a heuristic solution method is described.

3.1 Used data

To conduct analysis of the input parameters of the model, many different model instances were created on slightly different data. The preference matrix of the students ([illegible]) as well as the number of planned courses ([illegible]) per student was randomly generated for each instance. Furthermore, the capacity (a.k.a. [illegible]) of each course was drawn randomly from a range defined by [illegible] (lower bound) and the number of students (upper bound). The penalty that applies when an additional slot is used (ὴ Ὠ ὲ ῷ ὰ ὸ ώ Ὀὠ), the number of additional slots per course (ὲ ῷ Ὀ ὼ ὸ ὶ ὡ), as well as the minimum number of attendees per course (ά Ὠ ὲ ὂ ὸ ὸ Ὠ) are data parameters which are specified for each instance, but do not differ inside one instance.

The basis of the following analysis is a dataset including over 400 instances where each instance is the result of one model run. Beforehand, 21 different sets of parameter combinations have been defined to detect potential relationships. The model was solved 20 times for each of the 21 parameter sets. This ensures that the model does not work by accident, and reduces the randomness of our results, since the instances can be aggregated by the mean, which stabilizes the results and removes outliers. Each instance was solved on data for 200 students and 20 courses. To better understand this pipeline setup, Figure 9 (see in the appendix) illustrates the process one parameter set goes through. The following table represents the different parameter sets used:

parameter set	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
minCourseCap	20	40	60	80	100	20	20	20	20	20	20	20	20	20	20	20	20	70	20	75	70
penaltyExtraSlot	10	10	10	10	10	5	20	40	70	10	10	10	10	10	10	10	10	10	50	10	50
nbExtraSlots	20	20	20	20	20	20	20	20	20	5	20	50	100	20	20	20	20	20	20	20	20
minAttendees	8	8	8	8	8	8	8	8	8	8	8	8	8	2	10	20	50	40	40	45	8

Set0 is the base set, sets 1 to 4 differ in the minCourseCap parameter (which resembles the lower bound of the random range of which ά ῷ ὼ ὂ ὸ ὸ Ὠ is drawn), sets 5 to 8 differ in the penalty applied for extra slots. Set 9 to 12 adjust the number of additional slots and sets 13 to 16 modify the minimum number of attendees threshold. The sets 17 to 20 serve as random combinations to the data. With such setup we can isolate the effect of one parameter on the model.

3.2 Analysis of parameters

In the following, the effect of the individual parameters on the model is analyzed to better understand how it works and how we can improve the allocation of students to courses. The objective function value was divided into its 3 components to better isolate the impact of parameters on the model. Pref_sum resembles the sum of all preferences of all students. If its value is low, most of the students were assigned to courses of their preference. Penalty_nbCourses resembles the penalty which is added to the objective function if a student is assigned to less courses than he requested. If this value is high, many students were not assigned to their wished number of courses. The component penalty_addSlots contains the penalty if additional slots were needed. If this value is high, many of the courses were overbooked, meaning extra slots were added.

$$\textbf{pref\_sum} = \sum_{i\in I}\sum_{j\in J} preferences_{ij}x_{ij}$$
$$\textbf{penalty\_nbCourses} = \sum_{i\in I}(nbPlannedCourses_i - \sum_{j\in J} x_{ij})M$$
$$\textbf{penalty\_addSlots} = \sum_{j\in J} y_j penaltyExtraSlots_j$$
$$\textbf{obj} = pref\_sum + penalty\_nbCourses + penalty\_addSlots$$

3.2.1 Sensitivity analysis using OAT approach

One-at-a-time (OAT) approach is used to analyze the effect of one parameter on the model. By keeping all parameters constant and changing only a single parameter, one can try to detect trends in the data. This effect is graphically visualized by plotting the objective value on different values of the selected parameter. With such a graphical analysis one can already see rough trends and correlations in the data. The dataset was averaged by parameter set, and additionally scaled to the range [0,1], to ensure that the single graphs are comparable with each other. For each of the following parameters, the average of the 20 instances of each of the 5 parameter sets considered was formed and plotted (e.g., for MaxAttendees set 0 to 4).

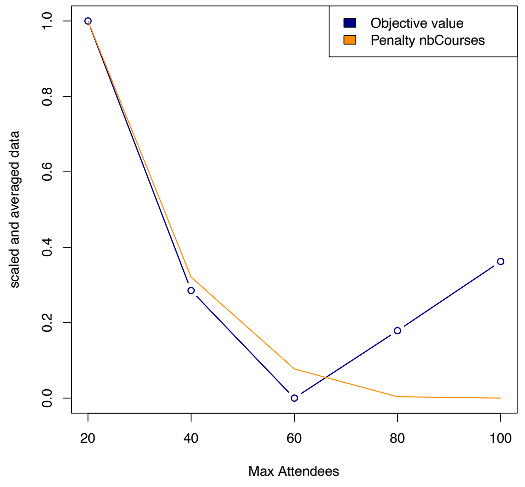


Figure 1 ï MaxAttendees; parameter sets 0 to 4

Starting with the parameter MinCourseCap, which resembles the lower bound of the range from which the MaxAttendees parameter is randomly drawn. We expect the model to profit from larger courses (more course capacity *maxAttendees$_j$*). It becomes clear that the parameter is optimal approximately at the value 60, where the objective function is minimized (see Figure 1). Here it seems as if a quadratic relationship between the objective value and MaxAttendees could be present. It is also evident that the penalty for undercutting the desired number of courses (orange line) decreases strictly monotonically with increasing MaxAttendees (. As the course capacity maxAttendees increases, students are more likely to get a slot in a course.

Next, the MinAttendees (ά ῞Ω ὲ ö ò ò Ὶ parameter is considered. It controls that a certain minimum number of participants are assigned to a course. Looking at the graph (see Figure 6 in the appendix) it is obvious that a higher minimum number of students leads to a higher objective value as well as to a higher penalty_nbCourses: the model must assign more students to courses of low priority, since surpassing the value for minAttendees ىά ῞Ω ὲ ö ò ò Ὶ is a hard constraint. More students are assigned to a lower number of courses than desired, which has a negative effect on our objective function. MinAttendees should therefore be chosen as low as possible.

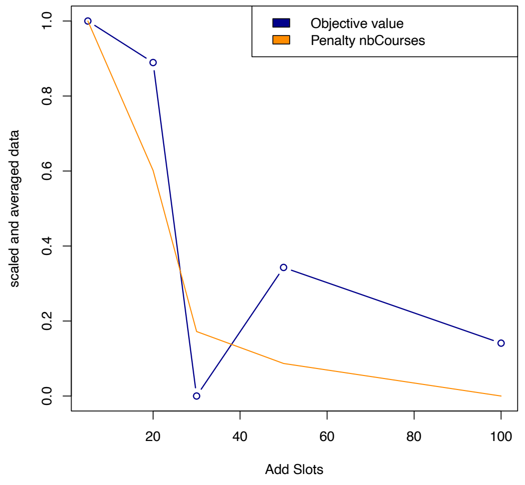


Figure 2 ï AddSlots; parameter sets 0, 9, 10, 11 & 12

The parameter AddSlots, describes the absolute number of additional slots per course, which can be booked for an additional penalty on the objective function. The slope of the curve is not quite clear (Figure 3). However, it seems as if a higher number of AddSlots generally has a positive influence (in the sense of minimization) on our objective function. In addition, the minimum is at a value of 30 additional slots. penalty_nbCourses also seems to decrease with increasing AddSlots, especially strongly up to the threshold value of 30. To optimize our model results, one should choose the parameter AddSlots as high as possible.

Next, the parameter AddSlotsPenalty is considered, which measures the amount of penalty if an additional slot is needed for a course. The curve shows that the minimum is at an AddSlotsPenalty of 20 (see Figure 7) and the objective values and the values for penalty_nbCourses increase with increasing values for

AddSlotsPenalty. Therefore, the model responds negative to a higher AddSlotsPenalty parameter, which should optimally set as low as possible.

Finally, the parameter M is evaluated. As already described in Extension 1, M is the factor by which the delta between the number of courses scheduled for a student and the number of courses actually taken is penalized. Thus, M has a direct influence on penalty_nbCourses. The selection of the value of ü is crucial for the model. If the value of ü is too small, then it might be reasonable to allocate only a few courses. If the value of ü is too big, then this soft constraint might never be violated, and each course is allocated. As a result, the value for ü must be big enough to guarantee that students are able to visit the desired lectures. Looking at Figure 8 in the appendix, the curve shows that penalty_nbCourses decreases as ü increases, which seems paradoxical at first. In fact, however, at high value for ü the optimization model is forced to assign as many students as possible to their desired number of courses. Thus, the penalty decreases. The threshold point here seems to be 20, which is the number of courses. If ü حى ς ȷπit is better for the model to assign a student one less course than to assign him a low preferred course. If ü جي ς ȷπthe model would rather assign a student to their least preferred course (rank 20) than not at all.

3.2.2 Sensitivity analysis using regression approach

Another method to analyze parameter sensitivity is a regression approach. Here, the estimated parameters can be interpreted as effect strength or sensitivity of the respective parameter: they show how much a change in one parameter effects the response variable e.g., the objective value. In contrast to the OAT approach, the regression approach does not require the dataset to be split. In the following section, the best (in terms of various model KPIs such as Black Information Criteria and adjusted R2) regression model was created for each of the 3 components of the objective function using general-to-specific methods. The model selection did not focus on predictability, but on how well the model can explain the data.

First, however, the correlation matrix of all feature and target variables is considered (see Figure 3). Here, the correlation is calculated using the Pearson correlation coefficient. One can clearly see how the 3 components of the Objective function (pref_sum, penalty_nbCourses and penalty_addSlots) correlate positively with the

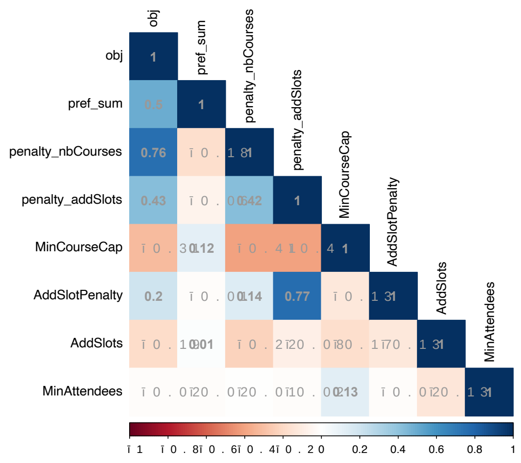


Figure 3 ï Correlation plot of features and targets

objective value as expected. MinCourseCap seems to have a negative influence on the objective. The positive correlation between AddSlotsPenalty and penalty_addSlots is also clear. AddSlots and MinAttendees, on the other hand, hardly seem to correlate with the objective and pref_sum does not seem to correlate with any of the parameters.

This is also confirmed by a regression analysis with pref_sum as the response variable (see Output 1 in the appendix): the adjusted R2 for the best model is 0.021 (corresponding p-value of the F-statistic = 0.0147). None of the regression parameters have a significant (level of significance: 5%) effect on pref_sum. This indicates that the variation in pref_sum cannot be explained by our estimated regression parameters but arises from other unknown influencing factors or is completely random. Here, the question arises whether the optimization model addresses student preferences at all or assigns courses completely randomly. The latter thesis can be rejected by comparing our solution with an optimal solution (see

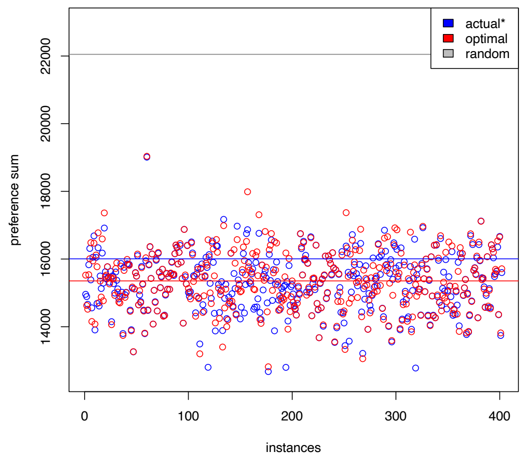


Figure 4 - The lines indicate the mean of the data. Note that for the actual data mean (blue line), the mean penalty_nbCourses was added to account for not meeting all students demands. Theoretical derivation of the optimal and random data see Formulation 1.

Figure 4): it turns out that our optimization model performs almost as well as the optimal preference distribution and significantly better than a random distribution (see Formulation 1 in the appendix for the theoretical test setup of optimal and random distribution). This means that our model is always able to successfully assigns students to their preferred courses. It seems that none of the model parameters maxAttendees (ὒ ώ ὼ ὃ ὸ ὸ Ὠ ὲ Ὼ minAttendees (ά Ὢ ὲ ὃ ὸ ὸ Ὠ,ὲ Ὼ Ὼ Ὼ nbExtraSlots [illegible] on penaltyExtraSlot [illegible] has such a [illegible] large effect on the model that it cannot be compensated by the other objective function components.

Now we take a closer look at the component penalty_nbCourses, which shows if the model is able to assign students to their planned number of courses. We set up a regression model with penalty_nbCourses as the dependent variable: (see Output 2). The best model achieves an adjusted R2 of 28.7%, which is still not a very good fit, but sufficient to interpret the regression parameters. The parameter AddSlots has the largest influence with an estimated beta of -59.58. Thus, when a course receives an additional slot, penalty_nbCourses decreases on average by almost 60, meaning that less students are assigned to a lower number of courses than requested. The parameter MinCourseCap also has a significant negative influence on penalty_nbCourses. On the other hand, AddSlotPenalty and MinAttendees seem to have a positive (significant) impact on the response variable: increasing the penalty for additional slots and increasing the minimum number of attendees per course therefore results in a higher number of students not being assigned to the wished number of courses. These findings are consistent with the OAT analysis.

Lastly, a regression model for penalty_addSlots (see Output 3) is set up. This model is used to explain how the parameters effect the overbooking of courses. The best model can explain the variation in the data well with an adjusted R2 of 68.4%. We have two significant regression parameters, MinCourseCap and AddSlotPenalty. AddSlotPenalty has the largest effect with an estimate of 6.29. This seems plausible since AddSlotPenalty is part of the definition of penalty_addSlots. Compared with the average number of overbooked courses (on average 7.35 of 20 courses have been overbooked) this indicates that an increase in AddSlotPenalty of 1 unit, has an impact of less than 1 (~0.86) per overbooked course on the component penalty_addSlots. The model tries to avoid overbooking with increasing penalty. The second significant parameter is MinCourseCap, which has a negative effect on the dependent variable. Enlarging the number of seats in a course reduces the overbooking of a course.

3.2.3 Analysis of model constraints

To understand the r e s t r i c t i o n s o f o u r m o d e l , t h e s l
constraint is plotted in the following boxplot (Figure 5). It shows the (absolute) aggregated slack of the 3 constraints in our model. One can clearly see that constraint 1 (nbPlannedCourses) has the least free capacity with a mean of 34.08, whereas the other hand, constraint 3 (MinAttendees) has the most free capacity with a mean of 1754.05. Hence, constraint 1 constrains our model the most and should be the first starting point for a possible improvement.

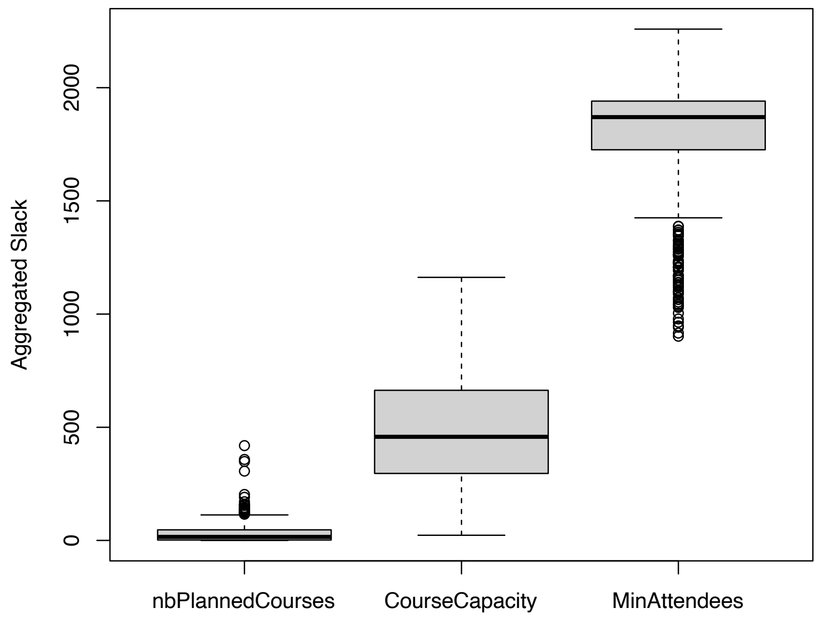


Figure 5 ï Boxplot of slack of each constraint

3.2.4 Conclusion

Like pref_sum, the regression analysis on the objective value does not have a good explanatory power (adjusted R2 of 16.87% see Output 4) and was therefore not explained in detail. However, since the objective value is by definition related to all 3 components in the form:

ɛ ὣ῀Ωἡ ὶ Ὠῖὣ ௲ ἡ Ὠ ὲ ὣ῀ἂ ὣ῀ ὣ῀ ɛ ό ௲ ἣ Ὠ ἅ ὣ῀ἂὩὩ῀Ὑἱὰ ɛ ὸ

the individual influencing factors can be directly transferred to the objective value. But why is this not evident in the regression model of the objective value? It turns out that the 3 components of the objective value are proportionally different in size: pref_sum accounts for 94.74% of the objective value on average, whereas penalty_nbCourses accounts for just 4.44% and pref_addSlots for 0.82%. The largest proportion cannot be explained (see Output 1) and therefore not to be estimated by the regression model, the other two components are lost i n t h i .s ñ n o i

Nevertheless, we can now combine the findings of the individual models and multiply the parameters by the appropriate weights:

ύ Ὼ ὝὩὨὩ Ὼ ὲ ὣ ἃὣ ὣ ἑ ί Ὼ ππ τ τ σ υ π φ φ

ύ Ὼ ὝὩὨὩ Ὼ ὲ ὣ ἃὣ ὣ έ ό πὩ Ὼπί ψ ς ς ψ υ ρ σ

	penalty_nbCourses		penalty_addSlots		obj
	estimate	weighted	estimate	weighted	
MinCourseCap	-19,0585	-0,845257054	-1,9374	-0,015941727	-0,861198781
AddSlotPenalty	11,2573	0,499268685	6,2891	0,051749312	0,551017997
AddSlots	-59,5802	-2,642421193	0	0	-2,642421193
MinAttendees	11,7424	0,52078319	0	0	0,52078319

Consequently, the parameter AddSlots has the greatest influence on our objective value. When weighted, one additional slot decreases our objective value on average by 2.6454 and so on. The following actions should be recommended to the school/university administration if student sectioning is to be improved: the largest impact has the increase of additional slots for each course. If applicable, this is where improvement should be started. Second, the overall course capacity should be increased. Lastly the minimum number of required attendees should as well as the penalty for additional slots (if applicable) should be lowered.

3.3 Heuristic approach to the problem

3.3.1 Reasons for a heuristic

Since an exact solution of the problem is very time consuming for large data sets an approximate solution that is simpler to calculate is often a good alternative that suffices in many instances. The method of approximating the optimal solution is called a heuristic. It is designed to reduce the complexity of the problem while generating a good solution.

In our heuristic we take into consideration the number of planned courses ὲ ὦ ὗ ὰ ὥ ὲ ὲ ῼ ῼ, the minimum capacity ὒ ῼ ὲ ὂ ὸ ὸ ῼ of a course ῼ and the preferences ὴ ὶ ῼ ῼ ῼ ἴ of the students ῼ Thus, it is a simplified problem because additional slots ὲ ὦ Ὸ ὼ ὸ ὶ ί are not considered.

3.3.2 Pseudocode

```
array sum_preferences[j]                                                                                     (1)
while preferences[i][j] != null do                                                                           (2)
        calculate the sum of preferences for each course and save the results in a new array sum_preferences[j]   (3)
end                                                                                                          (4)
sort sum_preferences[j] in an increasing order                                                               (5)
int nbPlannedCourses[i] = new Array(); //contains the number of planned courses for each student             (6)
while (nbPlannedCourses[i] != null && sum of nbPlannedCourses[i] > minCapacity[j] && sum of nbPlannedCourses[i] < maxCapacity[j]) do  (7)
      open course with highest popularity  minimum of sum_preferences[j]                                     (8)
      nbPlannedCourses[i] --                                                                                 (9)
      objective value += preferences[i][j] * x[i]                                                            (10)
end                                                                                                          (11)
```

In lines 2-4 we iterate through the courses and students and calculate the total sum of preferences for each course. We assume that the ranking is strictly ordinally scaled. This means that each course has received a ranking by a student. We know that a small value for a preference implies a high preference to attend the course. By building the total sum, we know which course is ranked highest and lowest by all students. The result is an array which contains of the total sums of preferences for each course.

Now, we want to internally rank the courses based on their popularity. We start by sorting the total sum of preferences in an increasing order in line 5. This transforms our array so that we can access the courses of decreasing popularity (and increasing value). In the following, we want to open courses while maximizing the courses ĝpopularity. In a similar manner, we build the sum of the planned courses for each student in line 6.

Now we open courses if certain conditions apply. We open another course if there are students which still want to attend courses. We obtain this information by accessing the array which contains of the sum of planned courses. If each element in the array is null, then no further course should be opened because the students now visit as much courses as they intended to. Furthermore, the course which is being opened must respect the minimum and maximum capacity of the course. So, the number of courses that still have to be allocated must be greater than the minimum capacity and smaller than the maximum capacity.

If this is the case, then courses can still be opened. The course which is opened first is the one with the highest popularity (line 8). Of course, we prioritize those students who have ranked this course the highest. When the most popular course is already opened, the second most popular course is opened. In the following, we decrease the number of courses a student still wants to attend because we have just allocated the student to a course. Now, we calculate the objective value. It is calculated like in the optimal objective function of the main model. We multiply the value for the preferences with ὼ̤ .ج If multiple courses are opened, this value is added on the prior objective value.

3.3.5 Runtime analysis of the pseudocode and the optimal solution

To compare the runtime of the heuristic and the optimal program, we will first identify the runtime of our heuristic. Then, we will use a generic approach to show that the pseudocode is faster than the optimal solution.

Let $F(n)$ be the runtime of our optimal solution.
In our objective function, we have $F(n) = O(n^2) + O(n) + O(n)$
After all results have been calculated, the minimum among the results is chosen.
Thus, we have to go through each result once and compare it with the other existing results in order to see whether it is the minimum of all feasible result, which takes $O(n)$.
Since this has to be done for each statement, we have a nested statement once more and
$F(n) = O(n) * O(n^2) + O(n) + O(n) = O(n^3)$

Now, we will show the runtime of our heuristic.

Let $T(n)$ be the runtime of our heuristic.
$$T(n) = O(1) + O(n^2) + O(nlogn) + O(1) + O(n) + 3*O(1)$$
$$T(n) \in O(n^2)$$

In line 1, a new array is initialized. Line 2-4 is a nested while loop, which results in a quadratic runtime. For sorting, HeapSort is used because it runs in O(nlogn) as for the worst-case time complexity and O(n) for the best-case time complexity. Since O(n) Ő O(nlogn) Ő O(n^2), O(n^2) is still the dominant term. In the following, the initialization of the array can be completed in O(1), the while loop requires linear time and in the while-Loop, three constant statements are run which can be neglected. In a nutshell, the runtime of our heuristic runs in O(n^2). This means that our pseudocode, which runs in O(n^2) is faster than the optimal program, which runs in O(n^3).

We [illegible] prove by definition that $O(n^2) \in O(n^3)$

Theorem $O(n^2) \in O(n^3)$

Proof

By definition, there exist two integers $n_0$and c such that $f(n) \leqslant c * g(n)$.

Here, $f(n) = O(n^2)$ *and* $g(n) = O(n^3)$.

Let n_0 1. Choose c = 2.

Thus, $f(n) = O(n^2) \quad 2 * g(n) = O(n^3)$ *for all* $n \geqslant 1$

Appendix

Figures

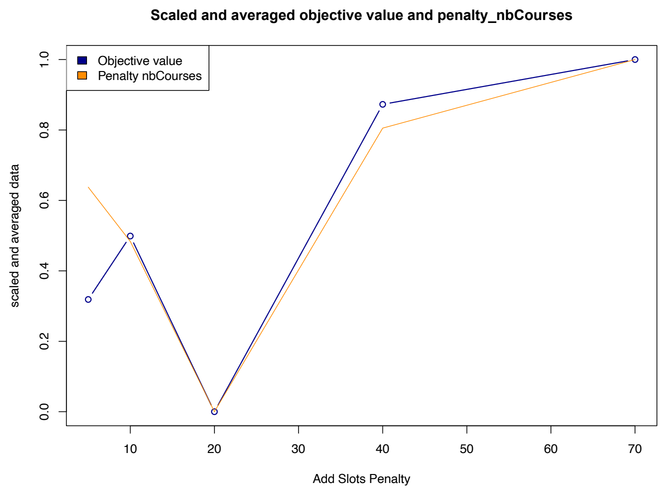


Figure 6 ï AddSlotsPenalty; sets 0, 5, 6, 7 & 8

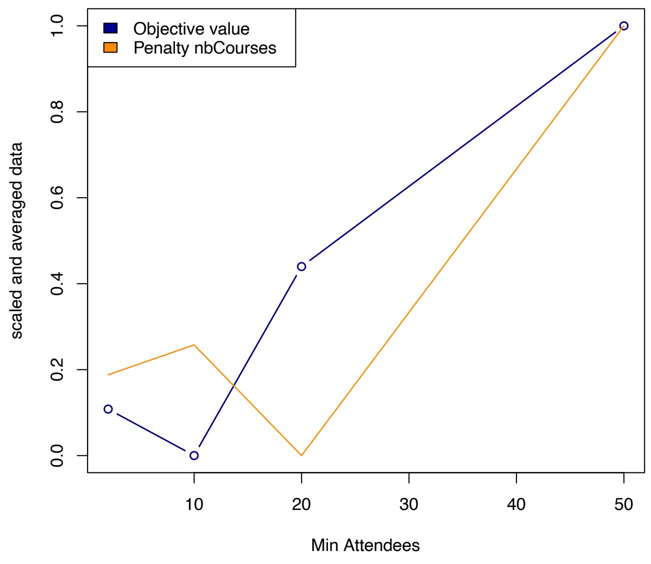


Figure 7 ï MinAttendees; sets 13, 14, 15 & 16

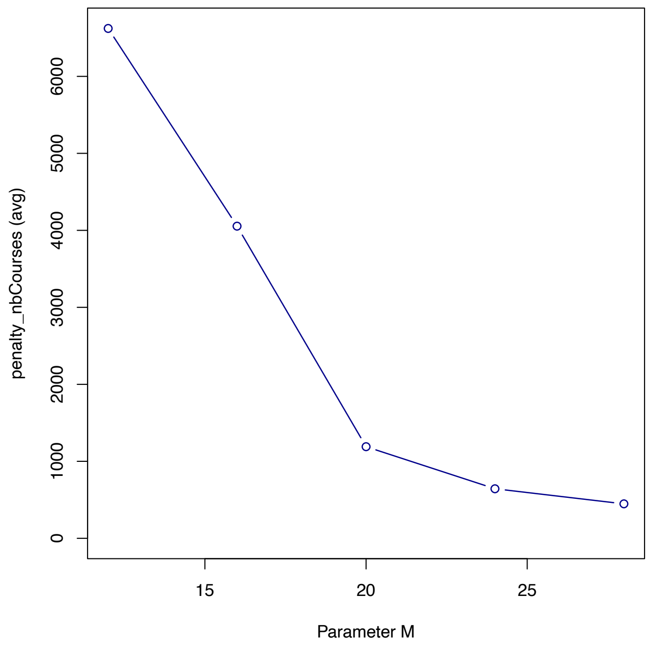


Figure 8 ï Parameter M

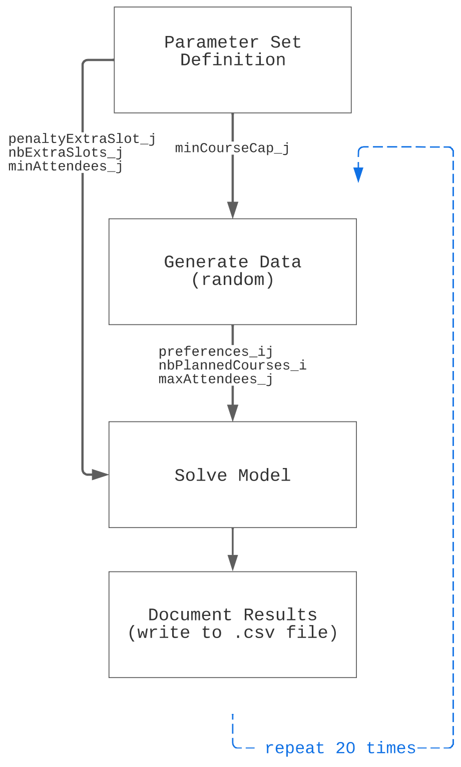


Figure 9 ï Flow chart of model pipeline for one parameter set

Mathematical formulations

$$\text{be } k_i = nbPlannedCourses_i \quad \forall i \in I,\ n = nbStudents \quad \text{and} \quad m = nbCourses$$
$$pref\_sum_{opt} = \sum_{i \in I} \frac{k_i(k_i - 1)}{2}$$
$$pref\_sum_{rand} = \sum_{i \in I} (\frac{1}{m} \sum_{j \in J} j)^2 = n \cdot E(J)^2$$
$$pref\_sum_{model} = \sum_{i \in I} \sum_{j \in J} preferences_{ij} \cdot x_{ij}$$

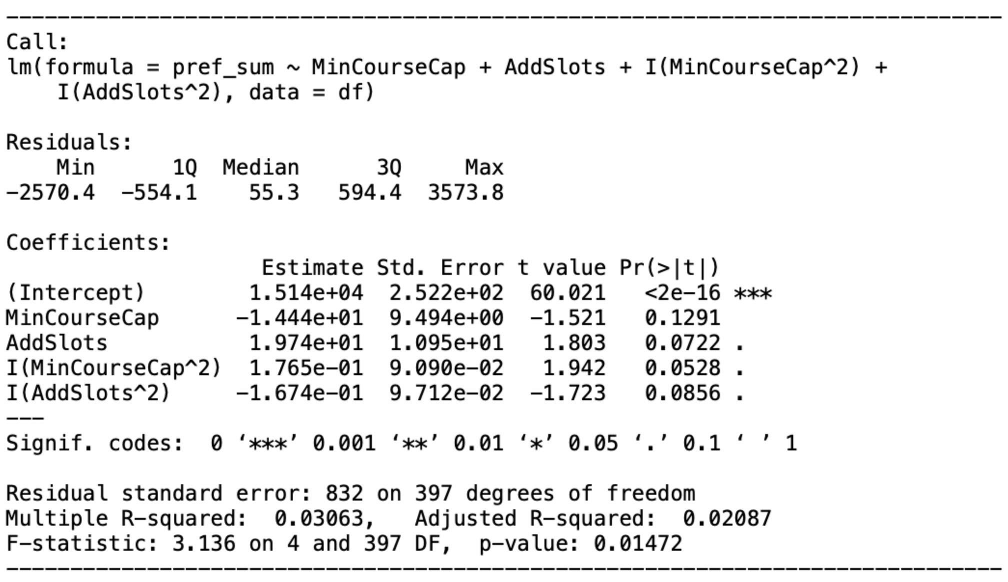
Formulation 1 - The optimal distribution is the best k courses, so the best possible outcome is the sum to the kth course. A random distribution is equal to the mean course rank multiplied by the mean number of nbPlannedCourses for each student. Both factors correspond to the expected value of a draw from all courses.

$$pref\_sum = \sum_{i \in I} \sum_{j \in J} preferences_{ij} x_{ij}$$
$$penalty\_nbCourses = \sum_{i \in I} (nbPlannedCourses_i - \sum_{j \in J} x_{ij}) M$$
$$penalty\_addSlots = \sum_{j \in J} y_j penaltyExtraSlots_j$$

$$\begin{array}{lll}
\text{Minimize} & obj = pref\_sum + penalty\_nbCourses + penalty\_addSlots & \\
\text{Subject to} & \sum_{j \in J} x_{ij} \leq nbPlannedCourses_i & \forall i \in I \\
& \sum_{i \in I} x_{ij} \leq (MaxAttendees_j + nbExtraSlots_j \cdot y_j) \cdot z_j & \forall j \in J \\
& \sum_{i \in I} x_{ij} \geq MinAttendees_j \cdot z_j & \forall j \in J \\
& x_{ij}, y_j, z_j \in \{0, 1\} & \forall i \in I, j \in J
\end{array}$$

Formulation 2 ï Mathematical representation of the optimization model; Extension 4

Outputs



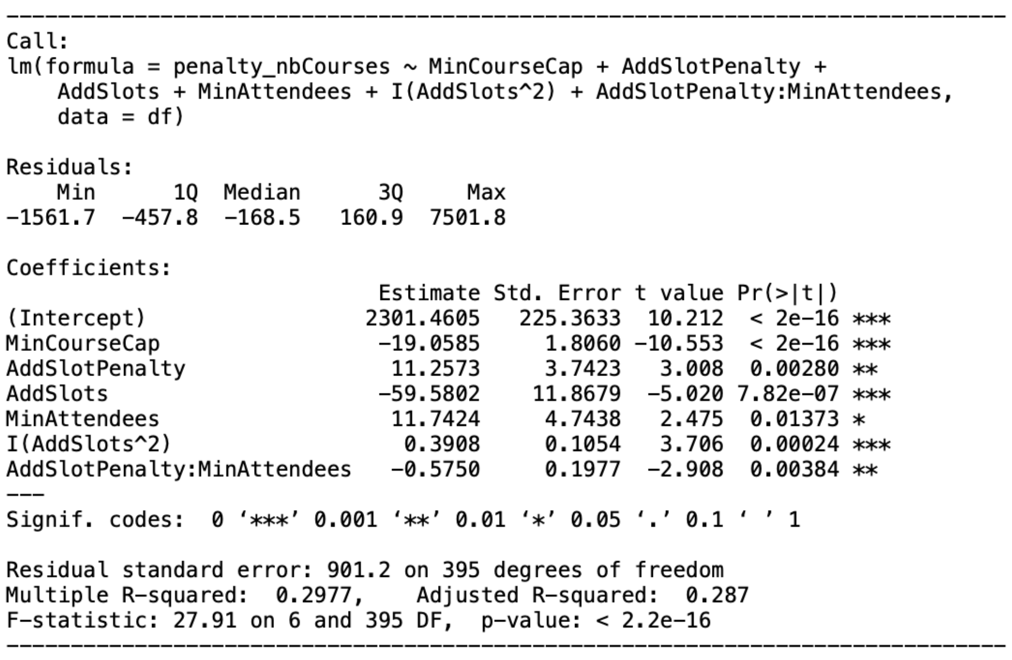
```
------------------------------------------------------------------------------
Call:
lm(formula = pref_sum ~ MinCourseCap + AddSlots + I(MinCourseCap^2) + 
    I(AddSlots^2), data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-2570.4  -554.1    55.3   594.4  3573.8 

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)    
(Intercept)        1.514e+04  2.522e+02  60.021   <2e-16 ***
MinCourseCap      -1.444e+01  9.494e+00  -1.521   0.1291    
AddSlots           1.974e+01  1.095e+01   1.803   0.0722 .  
I(MinCourseCap^2)  1.765e-01  9.090e-02   1.942   0.0528 .  
I(AddSlots^2)     -1.674e-01  9.712e-02  -1.723   0.0856 .  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 832 on 397 degrees of freedom
Multiple R-squared:  0.03063,   Adjusted R-squared:  0.02087 
F-statistic: 3.136 on 4 and 397 DF,  p-value: 0.01472
------------------------------------------------------------------------------
```

Output 1 ï Regression output of pref_sum as the dependent variable



```
------------------------------------------------------------------------------
Call:
lm(formula = penalty_nbCourses ~ MinCourseCap + AddSlotPenalty + 
    AddSlots + MinAttendees + I(AddSlots^2) + AddSlotPenalty:MinAttendees, 
    data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-1561.7  -457.8  -168.5   160.9  7501.8 

Coefficients:
                             Estimate Std. Error t value Pr(>|t|)    
(Intercept)                 2301.4605   225.3633  10.212  < 2e-16 ***
MinCourseCap                 -19.0585     1.8060 -10.553  < 2e-16 ***
AddSlotPenalty                11.2573     3.7423   3.008  0.00280 ** 
AddSlots                     -59.5802    11.8679  -5.020 7.82e-07 ***
MinAttendees                  11.7424     4.7438   2.475  0.01373 *  
I(AddSlots^2)                  0.3908     0.1054   3.706  0.00024 ***
AddSlotPenalty:MinAttendees   -0.5750     0.1977  -2.908  0.00384 ** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 901.2 on 395 degrees of freedom
Multiple R-squared:  0.2977,    Adjusted R-squared:  0.287 
F-statistic: 27.91 on 6 and 395 DF,  p-value: < 2.2e-16
------------------------------------------------------------------------------
```

Output 2 ï Regression output of penalty_nbCourses as the dependent variable

```
-------------------------------------------------------------------------------
Call:
lm(formula = penalty_addSlots ~ MinCourseCap + AddSlotPenalty + 
    AddSlots + MinAttendees, data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-276.69  -29.34    2.69   30.75  423.31 

Coefficients:
               Estimate Std. Error t value Pr(>|t|)    
(Intercept)     89.2300    13.2682   6.725 6.13e-11 ***
MinCourseCap    -1.9374     0.1740 -11.137  < 2e-16 ***
AddSlotPenalty   6.2891     0.2491  25.244  < 2e-16 ***
AddSlots        -0.3373     0.2430  -1.388    0.166    
MinAttendees     0.3392     0.3086   1.099    0.272    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 88.46 on 397 degrees of freedom
Multiple R-squared:  0.6872,    Adjusted R-squared:  0.684 
F-statistic:   218 on 4 and 397 DF,  p-value: < 2.2e-16
-------------------------------------------------------------------------------
```

Output 3 ï Regression output of penalty_addSlots as the dependent variable

```
-------------------------------------------------------------------------------
Call:
lm(formula = obj ~ MinAttendees + I(AddSlotPenalty^2) + MinCourseCap:AddSlots, 
    data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-2730.5  -780.8   -60.1   724.7  5293.7 

Coefficients:
                        Estimate Std. Error t value Pr(>|t|)    
(Intercept)            1.672e+04  1.298e+02 128.822  < 2e-16 ***
MinAttendees          -2.577e-01  4.063e+00  -0.063  0.94946    
I(AddSlotPenalty^2)    1.455e-01  4.737e-02   3.071  0.00228 ** 
MinCourseCap:AddSlots -8.041e-01  1.018e-01  -7.901 2.73e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1181 on 398 degrees of freedom
Multiple R-squared:  0.1749,    Adjusted R-squared:  0.1687 
F-statistic: 28.12 on 3 and 398 DF,  p-value: < 2.2e-16
-------------------------------------------------------------------------------
```

Output 4 ï Regression output of obj as the dependent variable