So, I’m trying to describe in my own words what my current level of understanding is. It’s really basic and top down trying to get from my level of understanding to what you’re teaching me so I don’t miss anything.

**Defining the problem**

As always, defining the problem correctly is half the problem solved. This is also the reason why I place such a premium on understanding the rough solutions space to be able to help that definition. I now also see how impactful extra variations are because of the factorial nature of scheduling problems.

So, in defining the problem we need to see first creating the starting choices. Then we need a machine that generates possible solutions, then we need a process of eliminating solutions that we already know are ‘illegal’ (i.e. a class taught by a teacher that does not work that day or the same subject twice in a day). Then we need to apply a function the assigns some kind of ‘fitness’ score to the solution. This starts with simple functions, for instance “the number of students which have a double booked class” (which needs to be minimized) or “total spread of the lessons over the week” and ends with more complicated functions where multiple of such calculations will be combined using a weighing system. For the solving procedure this is relatively immaterial, apart from computing cost.

**Complexity of the problem**

A problem of this kind isn’t generally solvable by an easy step by step approach. This makes the first ‘go to’ answer to brute force it. However, given the number of variables, this is completely unfeasible. I make some quick calculations on the rough setup I’m proposing in the following sheet:

<https://docs.google.com/spreadsheets/d/1urO8Yk5bCbBzN71Gop6gK622ZhK_IUs10TW4nuuDNT8/edit?usp=sharing>

If we take lessons to be planned as the main factor, it turns out there are roughly 54 lessons to be planned. The simple approximation of creating all options is to pick one of 54 in the first, one of 53 in the second, etc. This would come down to 54! (factorial 54) which turns out to be 2,30844E+71. If we take a generous assumption of creating and testing one option in 1 millisecond, it would still take a sizable 7,32001E+61 years to try all options. Assuming teachers aren’t willing to wait for that answer, we need to do something else.

The roughest version would be to take the amount of time you wish to spend on the problem, say 10 minutes, and estimate the amount of time it takes for one combination to be tested (say 10 ms). This leaves us 60.000 tests out of the 2,30844E+71 to find the right answer.

This is where we are using an algorithmic approach to find the answer. We could just pick 60.000 random combinations, but this is unlikely to yield a satisfying result as we have explored less than 1/2,30844E+66 of the answer space.

**More formal description of the problem**

The needed parts are:

decision variables: describe our choices that are under our control;

objective function: describes a criterion that we wish to minimize (e.g., cost) or maximize (e.g., profit);

constraints: describe the limitations that restrict our choices for decision variables

In our model this would be:

Input:

[I’m not sure how to define these yet, and what level of normalization is required, it is clear that the complexity of the problem won’t allow for a simple ‘traveling salesman’ string or series of digits to define parts]

Plannable sessions as a list:

SessionID-Classroom-Day-Slot, such as 1-2-3 (monday, spelling, slot three) or 4-1-2 (thursday, math, slot 2)

Classroom-Subject

Teachers:

TeacherID-Subject Taught

TeacherID-Working on day

Groups:

GroupID-Subject-Level-Lessons needed

Students:

StudentID-Subject-GroupID

One output option offers the following for each session ID, in the example above, it would be 54 rows:

SessionID-TeacherID-ClassroomID-GroupID

(which with the tables above can be expanded with more metadata needed to calculate factors below)

After generating a number of these solutions we start elimination based on the constraints:

* Same group may never be on the same day
* Same teacher may not be in the same slot
* Teacher may only teach qualified subjects
* Student may not have more than x concurrent sessions

(This is where the choices start in definition. You could either define that concurrent sessions must be 0, must be below 5. You could also choose to include the total SUM of concurrent sessions is counted and used as an optimizing goal. In addition you could add addition weight where a student having 3 concurrent sessions is worse that three students with one concurrent session by calculations SUM([concurrent session for student]2)  
Of course you can combine them: reject anything with more than 3 concurrent sessions for one student, and optimize for smallest number of concurrent sessions)

Then we write a fitness functions which calculates one or a weighted combination of the following:

* Number of concurrent sessions per student
* Number of subjects that a teacher teachers
* Number of levels that a teacher teaches
* Spread of lessons over the week (3 lessons over 5 days is better than 3 days concurrently)

This function needs to yield a single fitness score that the algorithm can use to determine the ‘quality’ of the solution.

Again, lacking experience, I’m assuming that after building such an algorithm I’m assuming it’s a trial and error process to finetune the parameters of the fitness function as the first results come in and any human looking at them will hit their head and say “of course you can’t do x” which you then code into either the criteria or fitness function.  
  
In theory over the long run you could even have an algorithm optimize the algorithm. Make some of the variables and weights above variable themselves, ask a feedback score from the users and then feed that data into a training algorithm which might predict either a general optimal setting, or a better setting given the specific users.

**Black Box**

This is where the black box begins. You feed the decision variables, the objective function and the constraints into some sort of solver and then hit ‘go’ and it starts doing its work. The work of the designer and programmer is to familiarize yourself with the current state of possible algorithms and their pros and cons and select one based on problem size, need for parallelization, memory, computational size, etc.

**Genetic Algorithm**

This is where the suggestion of genetic algorithms comes in as an approach. We pick a number of solutions and then see which ones are reasonable fit (high score on objective function) and combine them and see if the combination/mutation is an improvement. By doing this over multiple generations we have a reasonable chance of finding an ever increasing fitness. In this we want to balance between breadth (number of concurrent solutions tested) and depth (generations). The more generations, the more optimization of a good formula, but we run the risk of running into a local maximum that does not represent a global maximum. The more concurrent solutions, the less change of a local maximum, but less optimization of the solution. The middle would be the square root, in this case (60.000 tests) would be about 250 (244,95). So 250 generations over 250 variances.

Undoubtedly, experience with these matters yields an optimal number of generations and variances and an expected processing time per variance.

In this case the main question would be how to encode the given solutions. It was planned for future sessions but I’d like to take a ‘common sense’ guess at it’s approach: In a way it seems obvious, given that we should search for the shortest, standard length approach we can find. In the case of the 54 variantions above it would be

SessionID-TeacherID-ClassroomID-GroupID

(where even session ID would be redundant because it would be deducible via the string order. If the number of sessions and groups don’t exceed 26 we would say:

SessionID-[A..Y] (for 25 sessions)

TeacherID-[1..3]

ClassroomID-[1..3]

GroupID-[a..z] (if max 26 groups)

So we could code one session as:

A11a (Group a is in classroom 1 with teacher one at moment A)

This then could extend into:

A11aB23bA32e... etc into a 216 character string (or a 324 digit number which is probably more efficient in a computer)

Then the genetic process could be applied to this string. The fitness function would take the string (and has access underlying metadata) as it’s parameters to calculate fitness.

**Integer Linear Programming**

An additional option seems to be an Integer Linear Programming where you enter the criteria into a ‘solver’ for which you can find libraries. Microsoft seems to support this too:

<https://msdn.microsoft.com/en-us/library/ff524508(v=vs.93).aspx>