Course Optimization

By Wyatt Blair

Problem Description

As a student, it can be hard to find time for every assignment in every class. When looking at a course syllabus, a busy student will try to determine which assignments in the class they should spend the most time on. In other words, the student has a total time allotment for a particular course and wants to find the distribution of that time which corresponds to the maximum grade.

Therefore, two implicit constraints are defined:

1. The sum of the distribution of times should equal the total time allotment
2. Each individual assignment grade must be a float within [0, 1]

These constraints are the borders of a closed N-dimensional tetrahedron in work-time space, ergo the Extreme Value Theorem guarantees that a solution exists in this closed, feasible region.

Problem Description–Building The Objective Function

First, I implemented Python objects called SchoolCourse, GradeSection, and Assignment. Each object is constituted by a collection of the smaller counterpart (i.e. a SchoolCourse is made up of a bunch of GradeSections and GradeSections are made up of a bunch of Assignments).

Each Assignment has a “runtime” or a total amount of time needed to obtain a 100% on the assignment. This runtime is sampled from a tightly grouped normal distribution, with a mean depending on the assignment type (e.g. some HW assignments take less time than others, but all take around 5ish hours). The function which determines the Assignment grade is quadratic, so that a sense of “momentum” through the assignment is represented.

The goal is to find the distribution of different times spent on different assignments which maximizes the overall course grade.

Optimization Model

$$\max_{\vec{w}} \quad F(\vec{w})$$

$$\text{s.t.} \quad k(\vec{w}) = 0$$

$$u_i(w_i) + t_i^2 = 0$$

Where,

$$F(\vec{w}) = \sum_g \frac{g_W}{g_N} \sum_i^{g_N} f_{g_{A_i}}(w_{g_i})$$

$$f_{g_{A_i}}(w_{g_i}) = \left(\frac{w_{g_i}}{g_{A_{ir}}}\right)^2$$

$$k(\vec{w}) = T - \sum_i^N w_i$$

$$u_i(w_i) = 1 - f_{g_{A_i}}(w_{g_i})$$

$\vec{w}$	is the work-time array, the ith element is the amount of time the user will spend on the ith assignment in the class
$F(\vec{w})$	is the objective function, representing the grade in the class.
g	is a Grading Section
g_{A_i}	is the ith element in the set of Assignments in the Grading Section
$f_{g_{A_i}}$	is the Grading function for the ith assignment in a Grading Section, evaluated with the ith work time in the Grade Section’s work times
$\frac{g_W}{g_N}$	is the ratio of the Grading Section’s weight to the number of assignments in the Grading Section
T	is the student’s designated total time allotment for the class
$u_i(w_i)$	is the upper limit function (upper limit is 1) of the ith Assignment’s Grade
t_i	is a slack variable, used for handling inequalities in Lagrange Multiplier Problems
N	is the total number of assignments

Optimization Model–Lagrangian Multipliers

Using Lagrange Multipliers, I obtain the following system of equations:

$$\nabla F(\vec{w}) - \lambda \nabla k(\vec{w}) - \sum_i^N \theta_i \nabla (u_i(\vec{w}) + t_i^2) = \vec{0}$$

$$k(\vec{w}) = 0$$

$$\forall i \in [1, N] : u_i(w_i) + t_i^2 = 0$$

This is a system of 2N + 1 equations, with 3N + 1 unknowns.
Those unknowns are:

- the values of w_i
- the value of λ
- the values of θ_i
- the values of t_i

I will have to consider the Complementary Slackness Conditions in order to reduce the number of variables in each equation.

Proposed Solution–Solution Space

There are 2^N possible permutations of the Complementary Slackness Conditions, where I either consider θ_i or t_i. Therefore, each case can be uniquely identified with a binary number. For example, if the Course has 3 total assignments, the different cases are:

	Variable Modifiers (Not their actual values)					
Case #	θ_1	θ_2	θ_3	t_1	t_2	t_3
0	0	0	0	1	1	1
1	0	0	1	1	1	0
2	0	1	0	1	0	1
3	0	1	1	1	0	0
4	1	0	0	0	1	1
5	1	0	1	0	1	0
6	1	1	0	0	0	1
7	1	1	1	0	0	0

To generate case #i, simply convert i to an N digit binary number (use leading zeros if needed). This is the Theta modifier array. To obtain the t modifier array, repeat the same process but start with N-i instead of i.

Note: By excluding N variables each time, I obtain 2^N sets of 2N+1 equations with 2N+1 unknowns, making each set theoretically solvable!

Proposed Solution–Analytical Solution

Rearranging the system of equations and taking the Complementary Slackness Condition into account, I obtain:

$$w_i = \frac{\lambda a_{i_r}^2}{\frac{g_{j_W}}{g_{j_N}} + \theta_i}$$

$$\theta_i = 0 \ \ \& \ \ t_i = \sqrt{(1 - \frac{\lambda a_{i_r} g_{j_N}}{g_{j_W}})(1 + \frac{\lambda a_{i_r} g_{j_N}}{g_{j_W}})}$$

OR

$$\lambda = \frac{T}{\sum_i^N \frac{a_{i_r}^2}{\frac{g_{j_W}}{g_{j_N}} + \theta_i}}$$

$$t_i = 0 \ \ \& \ \ \theta_i = \lambda a_{i_r} - \frac{g_{j_W}}{g_{j_N}}$$

Through inspection, one can see that there is a trivial solution when all θ_i values are set to zero. When θ_i is zero, λ consists only of constants and so can be directly calculated, meaning w_i also only consists of constants and ergo can also be calculated directly.

Of course it is only one solution out of the possible 2^N solutions, but it is certainly the easiest to calculate.

Proposed Solution–Limitations

Taking θ_i to be zero in all cases severely limits the number of solutions this model produces and in fact runs the risk of actually returning a *minimum*. In general, the theoretical best solution would be to solve each of the 2^N sets of $2N+1$ equations for every Slackness Complementary Condition case, evaluate the proposed solution, and take the solution with the best evaluation.

For the sake of time, I decided to just move forward with this single solution–however, better solutions likely exist in the solution space that are not obvious to me.

Validating Solution

Since I implemented the Course object earlier in the project, validating my solution was fairly easy.

There were a few attributes that the solution needed to have:

- All w_i values needed to sum to T.
- Each assignment grade needs to be a number between 0 and 1 (inclusive).
- The overall grade of the Course needs to be somewhat high.

I made a few instantiations of my Course object, tried optimizing, and inspected the results manually to ensure they all made sense.

Validating Solution–ExampleCourse

Example GradeSection instantiation

```
HW_section = GradeSection(
    assignment_type='HW',
    grading_weight=0.15,
    assignments=[
        Assignment('HW #1'),
        Assignment('HW #2'),
        Assignment('HW #3'),
        Assignment('HW #4'),
        Assignment('HW #5'),
        Assignment('HW #6'),
    ]
)
```

```
ExampleCourse = SchoolCourse(grade_sections=[
    HW_section,
    exam_section,
    quiz_section,
    attendance_section,
    project_section
])
```

Example SchoolCourse instantiation

Example Optimization Results for total Time Allotment of 2 days.

```
HW (20.1147%):
| HW #1: (Runtime: 5:00:15.582485) (Time Spent: 2:20:48.163911) (Grade: 21.9901%)
| HW #2: (Runtime: 4:04:53.936877) (Time Spent: 1:33:40.074262) (Grade: 14.6288%)
| HW #3: (Runtime: 4:58:49.614034) (Time Spent: 2:19:27.728797) (Grade: 21.7808%)
| HW #4: (Runtime: 4:52:37.303305) (Time Spent: 2:13:43.822965) (Grade: 20.8856%)
| HW #5: (Runtime: 5:04:00.006015) (Time Spent: 2:24:19.955672) (Grade: 22.5414%)
| HW #6: (Runtime: 4:38:04.789198) (Time Spent: 2:00:46.147374) (Grade: 18.8614%)
==============================
Exam (0.2403%):
| Midterm Exam 1: (Runtime: 2:32:43.110239) (Time Spent: 0:06:49.781936) (Grade: 0.2000%)
| Midterm Exam 2: (Runtime: 2:29:45.070133) (Time Spent: 0:06:34.012436) (Grade: 0.1923%)
| Final Exam: (Runtime: 3:15:43.833461) (Time Spent: 0:11:13.111103) (Grade: 0.3285%)
==============================
Quiz (0.0192%):
| Quiz #1: (Runtime: 0:13:25.823434) (Time Spent: 0:00:10.563930) (Grade: 0.0172%)
| Quiz #2: (Runtime: 0:13:45.075839) (Time Spent: 0:00:11.074738) (Grade: 0.0180%)
| Quiz #3: (Runtime: 0:13:45.343712) (Time Spent: 0:00:11.081931) (Grade: 0.0180%)
| Quiz #4: (Runtime: 0:14:59.039052) (Time Spent: 0:00:13.149306) (Grade: 0.0214%)
| Quiz #5: (Runtime: 0:15:01.361463) (Time Spent: 0:00:13.217329) (Grade: 0.0215%)
==============================
Attendance (8.0181%):
| Attendance #1: (Runtime: 1:26:02.310088) (Time Spent: 0:23:07.343125) (Grade: 7.2224%)
| Attendance #2: (Runtime: 1:28:22.341839) (Time Spent: 0:24:23.629507) (Grade: 7.6195%)
| Attendance #3: (Runtime: 1:33:41.943026) (Time Spent: 0:27:25.389003) (Grade: 8.5657%)
| Attendance #4: (Runtime: 1:26:33.715549) (Time Spent: 0:23:24.274569) (Grade: 7.3105%)
| Attendance #5: (Runtime: 1:24:45.512758) (Time Spent: 0:22:26.372420) (Grade: 7.0091%)
| Attendance #6: (Runtime: 1:27:08.099702) (Time Spent: 0:23:42.929662) (Grade: 7.4076%)
| Attendance #7: (Runtime: 1:35:29.547845) (Time Spent: 0:28:28.977760) (Grade: 8.8968%)
| Attendance #8: (Runtime: 1:28:19.643903) (Time Spent: 0:24:22.140439) (Grade: 7.6118%)
| Attendance #9: (Runtime: 1:38:04.804270) (Time Spent: 0:30:02.850670) (Grade: 9.3855%)
| Attendance #10: (Runtime: 1:36:51.099209) (Time Spent: 0:29:17.973362) (Grade: 9.1518%)
==============================
Project (57.0026%):
| Report #1: (Runtime: 12:51:10.342612) (Time Spent: 9:17:16.621999) (Grade: 52.2203%)
| Report #2: (Runtime: 14:57:33.964660) (Time Spent: 12:34:55.215163) (Grade: 70.7407%)
| Report #3: (Runtime: 12:19:42.904356) (Time Spent: 8:32:44.396632) (Grade: 48.0469%)
==============================
..............................
GRADE: 11.245%
TOTAL RUNTIME NEEDED FOR an A: 3 days, 21:21:36.119064
```

Key Insights/Major Conclusions

I have a suspicion that the solution I obtained through the simplifying assumption of setting θ_i to zero is in fact a minimum and not a maximum, as the optimizer appeared to be adding time into GradingSections with a low grading weight.

As such, the true solution to my problem is likely sitting somewhere in the 2^N-1 sets of equations I could not easily solve. Attempting to solve these equations through Sympy and other Python packages resulted in untenable runtimes, seeing as N is usually around 20 and the number of equations is 2N+1.

If I had more time, I would have attempted to implement a Gradient Descent algorithm using the same Gradient I had calculated for the Lagrange Multiplier problem. I could have used the Barrier Method to enforce my constraints.

Suggestions for Improvement–WorkCurves

I think a very important part of this problem is how one chooses to model the idea of a WorkCurve. In mathematical terms, that is the

$$f_{g_{A_i}}(w_{g_i})$$

function, for which I ultimately chose a quadratic function, to capture a sense of “momentum” throughout an assignment:

$$f_{g_{A_i}}(w_{g_i}) = (\frac{w_{g_i}}{g_{A_{i\,r}}})^2$$

However, I believe there are a number of different WorkCurves one could imagine, and perhaps different assignments use different WorkCurves. For example, a linear WorkCurve for a quiz or a hyperbolic tangent WorkCurve for a project. I experimented with different WorkCurves but it unfortunately obfuscates the math to a degree that eliminates the easily calculable solution I reported earlier in the slides.

Thank you for a great semester!