The Open Schedule

James E. Carson

University of Louisville

Table of Contents

[Introduction 3](#_Toc58075328)

[The Open Schedule 4](#_Toc58075329)

[The Flying Schedule 4](#_Toc58075330)

[The Resource—Pilots 5](#_Toc58075331)

[Similar Problems 5](#_Toc58075332)

[Applying the PuLP Optimizer to the Scheduling Problem 6](#_Toc58075333)

[Problem Linearization 7](#_Toc58075334)

[Defining the Problem within Python. 9](#_Toc58075335)

[Program performance 12](#_Toc58075336)

[Conclusion 13](#_Toc58075337)

[Future Work 13](#_Toc58075338)

[References 15](#_Toc58075339)

Introduction

The Schedule Sell- a weekly meeting generally on Thursday between the weekly scheduler, often called the *weekly*, and the Director of Operations, referred to as the DO. The weekly has spent days placing pilots for the following week’s schedule according to the DO’s priorities. The DO has certain upgrades and requirements he wants the squadron to accomplish for the next week and each pilot has specific qualifications and availabilities that are matched to different spots in the schedule. Now, the scheduler has to sell this complex matching schedule to the Director of Operations before the schedule can be signed and sent out to the squadron. Often, unrecognized problems in the schedule cause this meeting to draw out hours trying to reflow the entire schedule before it can be sent out to the squadron on Friday.

Scheduling pilots to flying missions continues to be a complex problem requiring copious man-hours to fulfill the desired priorities subject to numerous constraints. In this project, we apply the GEKKO Python optimization package to find a reasonable schedule that meets the required constraints.

Keywords: Scheduling, Optimization, Python, PuLP, Matching

The Open Schedule

## The Flying Schedule

A flying schedule is a map of sorties that are expected on certain time frame. A sortie is derived from a French word to indicate one mission or attack flown by a single plane. Sorties are generally grouped into missions which indicate multiple planes flying in concert as part of one overall mission. Generally, most missions are flown with four sorties and are numbered such that the #1 sortie is flown by the pilot leading the overall mission, the #3 sortie is flown by an *element lead* and provides a backup to the #1 pilot, and the #2 and #4 sorties provide direct support to the mission. Finally, missions are grouped into *go*’s, which indicate a group of missions flown over a similar time period. A typical weekly flying schedule will have morning and evening go’s everyday Monday through Thursday with a morning only go on Friday for a total of 9 go’s.

Generally included in the flying schedule are a few nonflying duties—generally referred to as *ground duties—*that require a specifically qualified pilot. Some examples of this include a supervisor of flying that monitors flying operations from the tower and a vault security officer that monitors the secure working are during flying operations. Ground duties like these and others may be shared between squadrons or required only on certain go based on the flying mission requirements.

This project avoids scheduling some of the more complex ground duties but instead focuses on the most periodic of these requirements—the operations supervisor. The operations supervisor is a specially qualified pilot that supervises the flying operations for that go. This ground duty is included in this project because there is always one such pilot and he or she is assigned for each go. This ground duty prevents a pilot from flying in that go, but generally still allows a pilot to fly or plan for subsequent go’s.

## The Resource—Pilots

Within the schedule, matching the resource of pilots can be somewhat challenging based on two primary constraints—qualifications and availability. Each pilot holds a number of different qualifications that define what positions he or she is able to fly. For the purpose of this project, we will focus on the four main flying qualifications: Wingman, Flight Lead, Instructor Pilot, and Flight Examiner. A Wingman(W) is the base flying qualification. This pilot can fly the #2 or #4 sortie as part of some mission. A Flight Lead (FL) is required to fly #1 or #3 sortie positions, but he or she can also fly in the #2 or #4 positions. An Instructor Pilot (IP) can fly in any sortie position and are also required in upgrade sorties which are used to increase a pilot’s qualification. A Flight Examiner (FE) provides flying check rides to other pilots on a biennial timeline to ensure a quality force standard. A general squadron breakup is approximately around 3:3:3:1 W:FL:IP:FE.

As with any personnel scheduling problem, one of the most important constraints is the availability and welfare of the pilot. Each pilot has some number of meetings or *commitments* throughout the week that prevents them from being available for certain go’s. Additionally, the mission preparation, brief, flying, and debrief provide considerable time requirements on each pilots schedule. Flying multiple days in a row is generally taxing on a pilots schedule and flying twice in one day is very difficult. Most pilots in this environment average around 8 or 9 sorties per month.

## Similar Problems

This process of matching pilots to available sorties within the schedule can be described as multiple iterations of the General Assignment Problem. In the General Assignment Problem can be described as follows: given there are a number of agents, tasks, and costs for each agent-task pairing, assign an agent to each task to minimize the overall cost. The General Assignment Problem is seen in many different fields that can maximize profits or minimize costs based on assigned assets to job requirements. Some examples of this problem include assigning workers to machines, teachers to classes, vehicles to routes, salesmen to contracts, and military vessels to patrol areas.

This problem is categorized in the NP-Hard subset of Computer Science problems indicating that there is no known solution that can be found in polynomial time (Papadimitriou & Steiglitz, 1998). Each go of the pilot scheduling problem is essentially its own General Assignment Problem, but the costs, or in this case the quality of schedule, is found by looking at a pilots schedule over the entire week. This weekly cost versus a daily assignment adds to the difficulty in finding an optimal solution—a quality schedule—to the pilot scheduling problem.

# Applying the PuLP Optimizer to the Scheduling Problem

In this project, we apply a Python LP modeler to handle the multiple specific optimization problem as it relates to the flying schedule. PuLP is a modeling suite for Python introduced in 2005 that specializes in Linear Programming (LP) and Integer Linear Programming (ILP) (Mitchell, et al., 2009). Linear Programming is an optimization technique used to solve mathematical problems that are defined by real variables subject to a linear optimization function and linear constraint equations. An example LP problem might be written as follows:  
An ILP require that the decision variables be integers, which can greatly change the solution that might have been found through an LP process. In order to apply a ILP solver, the pilot scheduling problem must first be simplified and expressed in a linear way.

## Problem Linearization

The main way that the pilot scheduling problem is fit into a an ILP format is through an expansion of variables. Traditionally, one might think of the problem variable as which pilot is flying in which line. This method makes difficult the task of ensuring the appropriate pilot qualification for each line as well as ascribing a metric for quality of schedule as it pertains to an individual pilot’s schedule. Instead, this project organizes a series of binary variables, also referred to as “indicator variables,” that represent every pilot either flying or not flying for every sortie (Sarkar, 2019). For this project, the binary variable called pilot\_status is created for each pilot for each possible sortie in the schedule. The pilot\_status variable shows 1 or *True* if a pilot is scheduled to fly in that sortie and 0 or *False* for when a pilot is not flying in that line.

For the example problem of 31 pilots, this creates 31 variables per line where only 1 variable can be true to indicate a specific pilot flying a specific line. With 64 sorties over the week, this indicator variable process creates 1984 binary variables within the project and defines a need for a higher number of constraints to prevent impossible solutions like two pilots scheduled to fly the same sortie. A similar process is also used to create a ground\_status variable indicating each pilots performance of the one defined ground duty above. As the ground duty is only once per go, this adds variables. These ILP simplifications allow the problem to be condensed for a PuLP solver but creates additional requirements in defining the real-world constraints.

To organize the problem in an ILP concept, three different spreadsheets are created to define real-world constraints. First, pilot qualifications are manually input into a spreadsheet on an increasing scale where 1 represents a Wingman and 4 represents a Flight Examiner. This allows an easy way for the optimizer to get at least a certain qualification for a sortie as higher qualified pilots can still fly lower qualification sorties. Second, a spreadsheet called go demand is manually created to define the qualification level required per sortie on the upcoming week schedule. Each sortie is generally referred to as a *line* indicating a line on the schedule. Third, pilot availabilities are simplified to a simple spreadsheet that provides a 1 or 0 for whether a pilot is available for a particular go. In this project, the table is manually created by looking at events a pilot has scheduled throughout the week. These simplifications allow the scheduler to work on a go-based timeline and views each go independent of associated timeline.

These three simplifications allow the scheduling problem to be reduced to three different lists of variables and constants which are referenced as dictionaries in the Python project application:

* + pilot\_status – an *n* x *s* list of binary variables indicating pilot *n* flying in sortie *s*
  + ground\_status – an *n* x *g* list of binary variables indicating pilot *n* is preforming ground duties during go *g*
  + pilot\_quals – an *n* list of pilots and their associated qualification level
  + line\_req – an *s* list of each line and the required qualification level defined by go (*g*) and line #
  + pilot\_avail – an *n* x *g* matrix of each pilot’s availability to fly in that specific go

being *n*  the number of pilots, *g*  the number of go’s, and *s*  the number of sorties

An ideal schedule allows pilots enough time to prepare for higher requirement sorties while also maintaining a similar level of sorties across all pilots. In equation form this might equate most closely to

## Defining the Problem within Python.

With a general scheme set-up, using PuLP for optimization becomes fairly straightforward. First, the three man-made spreadsheets are import into some array. In this project, a Pandas array is used to hold pilot\_quals, line\_req, and pilot\_avail arrays. Next, the variables, pilot\_status[*n* x *s*] and ground\_status[*n* x *g*], are created as dictionaries within the pulp framework with the following code: pilot\_status = pulp.LpVariable.dicts("pilot\_status", ((Line,PILOT) for Line in line\_req.index for PILOT in pilots.index ), cat='Binary').

Next, the model is created, and the objective function is set. For this application the Linear Program problem is set up as a Maximization problem via the code model = pulp.LpProblem("PilotSchedProb", pulp.LpMaximize). For this project application within Python, the objective function is defined off of increasing time between time intensive requirements. Within Python, this objective function declaration appears as follows:

model += pulp.lpSum(sum(sum(line\_req[go1,line1]\*pilot\_status[((go1,line1),pilot)]-  
line\_req [go2,line2]\*pilot\_status[((go2,line2),pilot)]  
for go1,line1 in line\_req.index for go2,line2 in line\_req.index if go2 < go1)  
for pilot in pilots.index))

Within the ILP model, the constraints help define a realistic solution by preventing impossible scheduling assignments. The following is a list of required constraints applied within the Python project application:

Table - Required Scheduling Constraints

|  |  |
| --- | --- |
| A qualified pilot must perform the required ground duty for each go | for i in range(1,10):  model+= sum(ground\_duty[i,x]\*pilot\_quals.loc[x,'TOP3'] for x in pilots.index) == 1 |
| Only one pilot can perform the required ground duty | for i in range(1,10):  model+= sum(ground\_duty[i,x] for x in pilots.index) == 1 |
| A pilot flying a specific line must meet the required qualification level | for line in line\_req.index: model +=sum(pilot\_status[(line,x)]\*pilot\_quals.loc[x,'QUAL'] for x in pilots.index) >= lines.loc[line, 'Requirement'] |
| Only one pilot can fly each line. | for line in line\_req.index: model += sum(pilot\_status[(line,x)] for x in pilots.index) == 1 |
| A pilot can only fly once within each go. | for pilot in pilots.index:  for go in range(1,10):  model += (sum((pilot\_status[((go2,line2),pilot)] for go2,line2 in line\_req.index if go2==go)) + ground\_duty[go,pilot]) <= 1 |
| A pilot cannot fly if he is unavailable to fly for that go. | for go,line in line\_req.index:  for pilot in pilots.index:  if(pilots.loc[pilot][go-1]):  model += pilot\_status[((go,line),pilot)] == 0 |

Because ILP simplification made some scheduling priorities difficult to code linearly, some scheduling objectives can be added to constraints. These constraints may take viable solutions out of the search space, but ideally maintain a better-quality solution space. If a schedule is unable to be found with these constraint in place, it may be removed to find some alternate solution. An example of a constraint in this category that was used in this project is flying a pilot in consecutive go’s. While a pilot may be able to fly on consecutive go’s, it is generally difficult to adequately prepare for the flight on the subsequent go. This constraint was added as follows:

Table - Desired Scheduling Constraints

|  |  |
| --- | --- |
| A pilot cannot fly in two consecutive go’s. | for pilot in pilots.index:  for go1,line1 in line\_req.index:  model +=pilot\_status[((go1,line1),pilot)]+sum(pilot\_status[((go2,line2),pilot)] for go2,line2 in line\_req.index if go2==(go1+1)) <= 1 |

With all of these constraints defined over all 2000 variables, writing the problem on single line 12-point font can take over 300 pages. We’ll spare including that ILP problem definition in favor of the above constraint definition tables.

With the variables, objective, and constraints defined, the final step is to run the model in PuLP. PuLP models the LP and allows multiple different solvers to be applied to the problem including GLPK, COIN-OR CLP/CBC, CPLEX, GUROBI, MOSEK, XPRESS, CHOCO, MIPCL, and SCIP (Mitchell & Roy, 2020). A PuLP model without a specific solver defaults to CBC or Coin-or Branch and Cut, which is an open-sourced mixed integer linear programming solver written in C++ (johnjforrest, et al., n.d.). Solving the model via the default solver is simply done via a model.solve() command. One nuance of PuLP is that it will not return an error in an unsolvable problem. Therefore, it is recommended to check that the solver found an optimum solution through a pulp.LpStatus[model.status] command. For the example problem described above, PuLP was able to find an optimum solution that meets all the given constraints.

The final objective of the project is to provide a useful output that can be used in the scheduling process. The final step of the code searches through all pilot\_status and ground\_status binary variables and checks the solver True selections. The pilot, go, and line number are then added to a list, sorted via go and line number, and added to a Pandas DataFrame for ease of display. Ideally, these printed lists can be presented to the squadron scheduler as an initial point to start creating a schedule. An example of this printed schedule output from the Python project is shown in Figure 1 and Figure 2 below.

Table

Description automatically generatedTable

Description automatically generated

Figure - Weekly Ground Top3 Schedule

Figure - Weekly Flying Schedule

## Program performance

As the objective of the program is to decrease time requirements for a scheduler to find an optimum schedule, the performance capability of the program is very important. While a large LP problem statement at first seems counterproductive, in practice, the solver is able to overcome the scale of the problem and determine a reasonable solution with surprising efficiency. The highest time requirement throughout the program rest with defining the objective function within PuLP with the large number of variables. An example run redefining the objective function over 50 different iterations showed an average time of 72 seconds. The time required to solve the problem, however, generally ran less than 0.25 seconds as seen as an average over 500 calls. The overall time requirement for the entire program is generally less than 2 minutes. This time can be greatly shortened by not defining a specific objective function. This would find a solution that meets all the required constraints but does not continue to goal search for any better solution. In practice, the addition of the objective function appeared to create better quality schedules that seems to also better equalize the number of sorties across all pilots.

When compared to a human designed schedule, the PuLP project still falls short in accomplishing the varied objectives that a DO would set. The first obvious shortfall is the ability to shift priority of sortie assignments based on current proficiency. When no dedicated priority is set, a general equalization of sortie count would also be desired. Second, a large part of designing the flying schedule is finding mission opportunities for pilot upgrades. This scheduling objective was not attempted within this PuLP project. Overall, this application could provide a descent starting point for a human scheduler to vary according to local objectives, but schedulers would be required to modify the given solution to create a true solution the scheduling problem.

# Conclusion

Overall, assigning agents to jobs within the pilot scheduling scheme continues to provide difficulty within the flying schedule and computer science in general. Using PuLP as an optimization platform can greatly decrease the time spent finding a beginning solution but is not completely suited to the intricacies involved with creating a flying schedule. The main shortfall in PuLP is an inability to balance multiple potentially conflicting scheduling optimization desires.

## Future Work

Further progression of the project can add methods for the objective function to encourage good schedules from the standpoint of a human observer. Some of these upgrades are mentioned below:

* Normalizing the number and types of sorties and sortie positions across pilots.
* Adding specific mission types or desired upgrades to available upgrades in the schedule.
* Maintaining a consistent early or late schedule for each pilot throughout the week.
* Giving priority to younger pilots to lead sorties and increase proficiency.

Progression down this problem statement will optimize situations where constrained resources and their allocation are directly related to the fitness of the resource allocation. As fields see this narrowing of cost margins, this multi-level optimization can provide better use of the resources and, hopefully, a little much-needed less stressful work environment.

References

johnjforrest, Vigerske, S., Santos, H. G., Ralphs, T., Hafer, L., Kristjansson, B., . . . Saltzman, M. (n.d.). *coin-or/Cbc: Version 2.10.5*. Retrieved from https://zenodo.org/record/3700700#.X8gqhxNKg1J: https://github.com/coin-or/Cbc

Mitchell, S. A., & Roy, J. S. (2020, Oct 22). *Project Description*. Retrieved Nov 2020, from PuLP 2.3.1: https://pypi.org/project/PuLP/

Mitchell, S., Kean, A., Mason, A., O'Sullivan, M., Phillips, A., & Peschiera, F. (2009). *Optimization with PuLP*. Retrieved November 2020, from PuLP 2.3: https://coin-or.github.io/pulp/index.html#

Papadimitriou, C. H., & Steiglitz, K. (1998). *Combinatorial Optimization: Algorithms and Complexity.* Mineola, New York: Dover Publications, Inc.

Sarkar, T. (2019, April 19). *Linear programming and discrete optimization with Python using PuLP*. Retrieved from towards data science: https://towardsdatascience.com/linear-programming-and-discrete-optimization-with-python-using-pulp-449f3c5f6e99