Developing Adaptive Pilot Schedule Generator in Private Aviation

Author: Owen Mudgett

Faculty Mentor: Dr. Mihhail Berezovski

Embry-Riddle Aeronautical University, College of Arts & Sciences

Industrial Partner: FlexJet

**Abstract**

Scheduling is an intricate problem that encompasses fields like nursing, first responders and pilots. A private company that flies a fleet of 270 jets is looking to improve their current schedule-generating process. This project aims to make an algorithm capable of fast generation of optimized schedules for the company’s pilots. The goal of such an algorithm is to have schedules that both meet work demands while also prioritizing preferences and happiness of the pilots being scheduled. Research on the topic of scheduling algorithms was placed on examples such as the nurse scheduling problem, as many techniques from these types of problems are very relevant to creation of an algorithm. Results of the algorithm will follow tour formats, total work days per bidding period, and other marked days such as training and mandatory off days. The algorithm should ideally give pilots more attractive schedules while optimizing schedules for the company’s needs.

**Introduction and Problem Statement**

The goal of this research project is to deliver an algorithm to optimize times of valid schedule generation for pilots. Pilot schedules are structured around tours, which primarily consists of a set number of days on work followed by a set of days off work. Scheduling is done during a “bidding period,” which is a 28-day period nestled between two 24-day periods. Ideally, in addition to tour scheduling, an algorithm should also be able to take in certain pilot preferences, such as number of total days worked during the “bidding period,” to make pilot happiness a priority. The primary focus for this problem is the actual schedule generation aspect. Not all valid schedules work, as pilot preferences are quite varied and one generated schedule would likely not satisfy all pilots involved. In terms of priority of preferences, the pamphlet provided by this project’s industrial partner detailing the problem has them listed as follows:

* Duty Work Days (Total days worked during period): 0
* Tour Preferences: 1
* Date Grouping On (Specific dates to work): 2
* Date Grouping Off (Specific dates to have off work): 3

To sufficiently meet company demands, the total amount of days worked by a pilot during each bidding period is prioritized above all other preferences. Other preferences, like seniority, can also be used to prioritize preferences from certain pilots.

Within the group assigned work on this project, this report will go into more detail in regards to sub ranking of tour preferences. A valid schedule with this method is within the range of acceptable days on (12-20), generated schedules in tours, and follows the selected sub-ranking lists of tours available for schedules.

**Methodology**

At the beginning of this project, a sample schedule was given which serves as an example for what the algorithm should be outputting. Due to the nature of the scheduling generation process, the example schedule can also be used as input to generate a new schedule, since the bidding period has a previous 24-day period that it must build off.

The example was provided by this project’s industrial partner, and includes labels for each day’s meaning for each pilot’s schedule. The ‘S’ label stands for scheduled, and denotes days that pilots are on-duty. As per the packet given at the beginning of the semester, the ‘M’ stands for mandatory off days, but those are not present in the final algorithm. There are several other day labels for days such as training, but those are also not included in the final algorithm.

The tours are drawn from default and preference sets, allowing the definition of both standard scheduling rules and additional preferred structures. The default and preference tour scenarios can be manually selected in the code by changing the arguments for the function set\_active\_tour\_set(int default\_set, int preference\_set). The combined set of tours are stored with ranking information, enabling weighted random selection that favors lower-ranked (higher priority) tours. The weights are given sequentially as the algorithm works down the selected list of tour preferences. The lists for both default tours and tour preferences are below:

*Default Tour Scenarios*

1. 4 | 3, 5 | 3, 6 | 4

2. 4 | 3, 5 | 3, 6 | 4, 7 | 4, 8 | 5

*Tour Preference Scenarios*

1. N/A

2. 4 | 4

3. 4 | 3, 4 | 4, 4 | 5, 4 | 6

4. 5 | 5, 6 | 5, 7 | 5

5. 5 | 5, 6 | 6, 7 | 7

6. 8 | 3, 9 | 4, 10 | 4, 11 | 5, 12 | 5

7. 4 | 3, 5 | 3, 6 | 4, 7 | 4, 8 | 5, 4 | 4, 5 | 4, 6 | 5, 7 | 5, 8 | 6

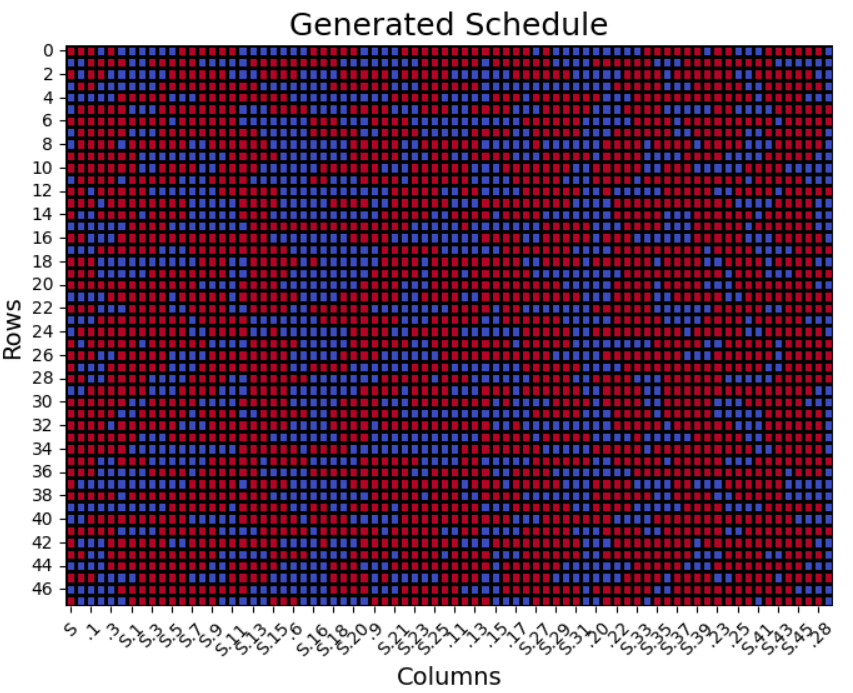
8. 14 | 14

For example, for list 6 of tour preferences, 8 | 3 is given sub-ranking 1 with a higher weight, whereas 12 | 5 is given the highest sub-ranking with the lowest weight of the list. All default tours are given a sub-rank of 0. The sub-rank determines the weight of a tour in the randomized tour selection process, as the weight is equal to 10 divided by the sub-rank (with default tours always having a weight of 10).

For each period, the program attempts to reach a randomized target number of workdays between 12 and 20 by repeatedly selecting valid tours that do not overlap existing workdays. A retry limit ensures that infinite loops are avoided in case the schedule becomes too crowded. This algorithm should result in schedules generated with non-overlapping tours that respect rest periods that follow tour workdays. The code for the program in C is provided in Appendix A.

**Solutions/Results**

Figure 1



This figure is a heatmap of one of the schedules generated by the algorithm. The schedule was run through a program to make this heatmap made by Jack Brooks, who is also working on this project. The heatmap is purely for visualization purposes and only includes days on (blue) and days off (red).

In terms of how effective this output is, it is possible and likely for any given schedule generated from the algorithm to be valid. While invalid schedules can be generated, a large majority of the schedules follow the validation criteria.

**Discussion/Conclusion**

Despite the schedule generation not being perfect, the program can generate schedules extremely fast, with generation of 270 simultaneous schedules completing in under five seconds. It is entirely possible, with an external validation program, for the algorithm to be ran multiple times in quick succession and have only the valid schedules kept in a timely manner.

The real-world impact from an algorithm like this will be felt by both the pilots and the company. The ability to generate multiple options and allow pilots to work around that is better for pilot happiness than constraining pilots to a single schedule that cannot be made in a timely manner. Such an algorithm also benefits the company, as optimized scheduling means greater probability that their pilot demand is met efficiently without sacrificing their pilots’ wellbeing.

**Acknowledgements**

We sincerely thank FlexJet LLC for giving us this project and for their support. Their help has been very important for our work and gave us a great chance to learn from real-world experience.

**Appendix**

Appendix A: Schedule Generation Algorithm in C

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define TOTAL\_DAYS 76

#define NUM\_SCHEDULES 270

#define MIN\_WORK\_DAYS 12

#define MAX\_WORK\_DAYS 20

#define MAX\_COMBINED\_TOURS 30

const int month\_days[] = {30, 31, 31, 31, 28};

const char \*month\_names[] = {"11", "12", "01", "02"};

void generate\_dates(char dates[TOTAL\_DAYS][6]) {

int day = 1, month\_idx = 0;

for (int i = 0; i < TOTAL\_DAYS; i++) {

sprintf(dates[i], "%s/%02d", month\_names[month\_idx], day);

day++;

if (day > month\_days[month\_idx]) {

day = 1;

month\_idx++;

}

}

}

typedef struct {

int work;

int rest;

} Tour;

Tour default\_sets[][5] = {

{ {4, 3}, {5, 3}, {6, 4} },

{ {4, 3}, {5, 3}, {6, 4}, {7, 4}, {8, 5} }

};

const int default\_set\_sizes[] = {3, 5};

Tour preference\_sets[][10] = {

{},

{ {4, 4} },

{ {4, 3}, {4, 4}, {4, 5}, {4, 6} },

{ {5, 5}, {6, 5}, {7, 5} },

{ {5, 5}, {6, 6}, {7, 7} },

{ {8, 3}, {9, 4}, {10, 4}, {11, 5}, {12, 5} },

{ {4, 3}, {5, 3}, {6, 4}, {7, 4}, {8, 5}, {4, 4}, {5, 4}, {6, 5}, {7, 5}, {8, 6} },

{ {14, 14} }

};

const int preference\_set\_sizes[] = {0, 1, 4, 3, 3, 5, 10, 1};

typedef struct {

int work;

int rest;

int rank;

} RankedTour;

RankedTour combined\_tours[MAX\_COMBINED\_TOURS];

int combined\_tour\_size = 0;

void set\_active\_tour\_set(int default\_set, int preference\_set) {

int count = 0;

for (int i = 0; i < default\_set\_sizes[default\_set]; i++) {

combined\_tours[count++] = (RankedTour){default\_sets[default\_set][i].work, default\_sets[default\_set][i].rest, 0};

}

for (int i = 0; i < preference\_set\_sizes[preference\_set]; i++) {

combined\_tours[count++] = (RankedTour){preference\_sets[preference\_set][i].work, preference\_sets[preference\_set][i].rest, i + 1};

}

combined\_tour\_size = count;

}

int pick\_weighted\_tour\_index() {

int total\_weight = 0;

int weights[MAX\_COMBINED\_TOURS];

for (int i = 0; i < combined\_tour\_size; i++) {

weights[i] = combined\_tours[i].rank == 0 ? 10 : (int)(10.0 / combined\_tours[i].rank);

total\_weight += weights[i];

}

int r = rand() % total\_weight;

for (int i = 0, sum = 0; i < combined\_tour\_size; i++) {

sum += weights[i];

if (r < sum) return i;

}

return combined\_tour\_size - 1;

}

void generate\_schedule(char schedule[TOTAL\_DAYS], char \*existing\_28day) {

memset(schedule, ' ', TOTAL\_DAYS);

// Map 28-day bidding period (days 29–52) into new schedule's days 0–23

memcpy(schedule, existing\_28day, 24);

// Define remaining two periods

struct { int start, end; } periods[2] = {

{24, 51}, // New Period 2

{52, 75} // Period 3

};

for (int p = 0; p < 2; p++) {

int days\_scheduled = 0, retries = 0;

int work\_target = MIN\_WORK\_DAYS + rand() % (MAX\_WORK\_DAYS - MIN\_WORK\_DAYS + 1);

while (days\_scheduled < work\_target && retries < 1000) {

int idx = pick\_weighted\_tour\_index();

int tour\_len = combined\_tours[idx].work;

int rest\_len = combined\_tours[idx].rest;

int range = periods[p].end - periods[p].start + 1;

int start = periods[p].start + rand() % range;

if (start + tour\_len - 1 > periods[p].end) {

retries++;

continue;

}

int conflict = 0;

for (int i = 0; i < tour\_len; i++) {

if (schedule[start + i] != ' ') {

conflict = 1;

break;

}

}

if (conflict) {

retries++;

continue;

}

for (int i = 0; i < tour\_len && days\_scheduled < work\_target; i++) {

schedule[start + i] = 'S';

days\_scheduled++;

}

}

}

}

void count\_period\_workdays(char \*schedule, int \*p1, int \*p2, int \*p3) {

\*p1 = \*p2 = \*p3 = 0;

for (int i = 0; i < TOTAL\_DAYS; i++) {

if (schedule[i] == 'S') {

if (i < 24) (\*p1)++;

else if (i >= 28 && i < 52) (\*p2)++;

else if (i >= 56 && i < 76) (\*p3)++;

}

}

}

int main() {

srand(time(NULL));

char schedules[NUM\_SCHEDULES][TOTAL\_DAYS + 1];

char dates[TOTAL\_DAYS][6];

generate\_dates(dates);

set\_active\_tour\_set(0, 0);

FILE \*input = fopen("GeneratedSchedule.csv", "r");

if (!input) {

printf("Previous schedule file not found.\n");

return 1;

}

char line[8192];

fgets(line, sizeof(line), input); // Skip header

for (int s = 0; s < NUM\_SCHEDULES; s++) {

if (!fgets(line, sizeof(line), input)) break;

char \*token = strtok(line, ","); // "Schedule X"

char prev\_segment[25] = {0};

for (int i = 0; i < TOTAL\_DAYS; i++) {

token = strtok(NULL, ",");

if (i >= 28 && i < 52) {

prev\_segment[i - 28] = token[0];

}

}

generate\_schedule(schedules[s], prev\_segment);

}

fclose(input);

FILE \*output = fopen("GeneratedSchedule.csv", "w");

if (!output) {

printf("Error opening output file.\n");

return 1;

}

fprintf(output, "Schedule ID");

for (int i = 0; i < TOTAL\_DAYS; i++) {

fprintf(output, ",%s", dates[i]);

}

fprintf(output, ",Period 1 Workdays,Period 2 Workdays,Period 3 Workdays\n");

for (int s = 0; s < NUM\_SCHEDULES; s++) {

int p1 = 0, p2 = 0, p3 = 0;

count\_period\_workdays(schedules[s], &p1, &p2, &p3);

fprintf(output, "Schedule %d", s + 1);

for (int i = 0; i < TOTAL\_DAYS; i++) {

fprintf(output, ",%c", schedules[s][i]);

}

fprintf(output, ",%d,%d,%d\n", p1, p2, p3);

}

fclose(output);

printf("Schedules generated successfully using prior 28-day period data.\n");

return 0;

}