Queen's UNIVERSITY

GRP_10: Flight Booking

Thomas Janna (18taj2)

Dumitru Cernelev (18pdc1)

Marc Brasoveanu (18mtb5)

Nathanael Yao (18ny13)

Course Modelling Project

CISC/CMPE 204

Logic for Computing Science

November 3, 2020

Abstract

Our model aims to represent a flight booking system for an airline on a given day. A number of pilots begin their day at the same airport and then through a series of flights throughout the day must end the day at unique airports with no two pilots on the same airport at any given time of the day. We are in the process of building our model to be valid under any amount of pilots and any number of airports.

In our model a timestep represents a time period of one flight. We assume that all flights to any airport take a single time step. Also pilots will have different max flight requirements for the day.

Propositions

PA_{ij} is true when pilot A has travelled to airports i then j (2 trips only)

PB_k is true when pilot B has travelled to airport k (1 trip only)

R_{AB} is a true when pilots A and B have made their trips for the day and ended at different airports

Code to create arrays which hold pilot paths. Rows represent timesteps and columns represent airports.

```
for timestep in range(N_TIMESTEPS):
    # Initialize pilot A variables
    airport_list = []
    for flight in range(N_AIRPORTS):
        flight = Flight(flight)
        airport_list.append(Var(flight))
    pilot_a.append(airport_list)

    # Initialize pilot B variables
    airport_list = []
    for flight in range(N_AIRPORTS):
        flight = Flight(flight)
        airport_list.append(Var(flight))
    pilot_b.append(airport_list)
```

Constraints

Our first type of constraint restricts a pilot and his/her plane from being at more than one airport on a single time step (i.e a plane cannot be at two places at once). To do this dynamically we build a string with logical operators and then use the python eval function to turn the string to

executable code to be given to the add_constraint method.

```
### Pilot A can only make one flight per timestep.
# Use a string to build the logical expression dynamically.
only_one_airport = ""

for timestep in range(N_TIMESTEPS):
    only_one_airport += "("
    for airport in range(N_AIRPORTS):
        # Current airport in loop is true.
        only_one_airport += "(pilot_a[%d][%d] &" % (timestep, airport)

        # All airports below airport are false, count up
        for count_up in range(0, airport):
            only_one_airport += " ~pilot_a[%d][%d] &" % (timestep, count_up)

        # All airports above airport are false, count up
        for count_up in range(airport + 1, N_AIRPORTS):
            only_one_airport += " ~pilot_a[%d][%d] &" % (timestep, count_up)

        # Remove the last & from the string
        only_one_airport = only_one_airport[:-2]
        only_one_airport += ") | "

    # Remove the last | from the string, use &'s to seperate timesteps
    only_one_airport = only_one_airport[:-3]
    only_one_airport += ") & "

# Remove the last & from the string.
only_one_airport = only_one_airport[:-3]
```

String to executable code:

$E.add\_constraint(eval(only\_one\_airport))$

We have implemented another constraint to ensure that only one pilot and their plane can be at an airport at any given
time. (i.e pilot A and pilot B cannot both be at airport 1 at the same time step)

```
### Pilot A and B can't be at the same airport at the same timestep (other than timestep 0).
for timestep in range(1, N_TIMESTEPS):
    for airports in range(N_AIRPORTS):
        E.add_constraint(~pilot_a[timestep][airports] |  ~pilot_b[timestep][airports])
```

Another type of constraint restricts a plane from staying at the same airport

for two time steps in a row. (i.e a plane cannot go from airport 2 to airport 2 after a time step) At the moment we have two constraints, one for each pilot since they have different criteria for max flights to make in a day. This will be migrated to one generic function.

```
### Pilot A cannot be at the same airport in two adjacent timesteps.
for timestep in range(N_TIMESTEPS - 1):
    for airport in range(N_AIRPORTS):
        E.add_constraint(~pilot_a[timestep][airport] | ~pilot_a[timestep + 1][airport])
```

```
### Pilot B cannot be at the same airport in two adjacent timesteps if they still have flights left to make denoted by N
for timestep in range(N_PILOT_B_MAX_FLIGHTS - 1):
    for airport in range(N_AIRPORTS):
        E.add_constraint(~pilot_b[timestep][airport] | ~pilot_b[timestep + 1][airport])
```

Pilot B must remain at airport if finished assigned flights.

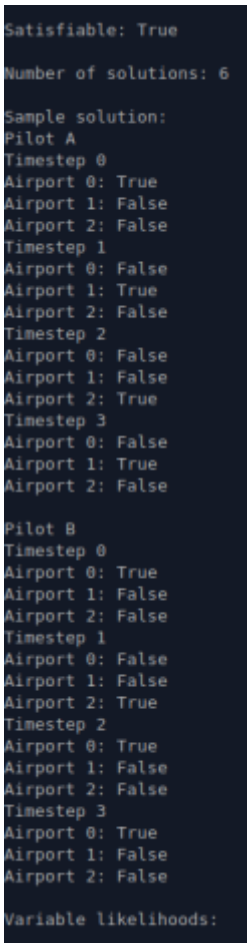
```
### Further, pilot B will remain at this airport after he is finished all its flights.
for timestep in range(N_PILOT_B_MAX_FLIGHTS, N_TIMESTEPS):
    for airport in range(N_AIRPORTS):
        E.add_constraint(iff(pilot_b[timestep - N_TIMESTEPS - 1][airport], pilot_b[-1][airport]))
```

Another constraint is that both pilot A and B start at airport 0.

```
### Pilot A and B both start at airport 0
E.add_constraint(pilot_a[0][0] & pilot_b[0][0])
```

Model Exploration

We tried testing our model by giving different numbers of airports, and time steps to check the number of solutions for each case and the order in which the pilots travelled.



We also tried adding constraints such as not allowing two pilots to travel to the same airport in a specific time step to see if the number of solutions or the order in which the pilots travelled were affected.

We explored giving different pilots a different number of required flights per day. We added this as a constraint and explored how it affected the number of possible solutions and the order in which pilots travelled that day.

First-Order Extension

The airline gets approached to expand their operations by one airport at a time but that airport must be travelled to at least once by one of the pilots in a day as well as every other consecutive airports added (Airports 4, 5 ... n)

Propositions: V(x) is true if a new airport can be travelled to at least once a day
A(x) is true if all airports have been travelled to at least once and Pilot A has another flight to make.

B(x) is true if all airports have been travelled to at least once and Pilot B has another flight to make.
Constraints: $\forall x.V(x) \rightarrow \exists x.(A(x) \vee B(x))$

Requested Feedback

Ideas on how to create variable number of pilots?

How can we use the concepts of a flight class to enhance our model?

How to convert output to a more readable form as stated in the propositions. Example: PA_{ij} is true when pilot A has travelled to airports i then j

Improve Jape proofs to help reach meaningful conclusions.

Jape Proofs Explanation

1. Idea: A constraint that makes sure that 2 planes can not be at the same airport. Where P(x) represents a function that determines if there is a plane at airport x. When P(x) is false that means there is no airplane at that airport, when true there is an airplane. Q(x) is a function that introduces new airplanes to airports. When Q(x) is true that means an airplane wants to go to that airport, when false no airplane wants to go to that location. We then created a disjunction between the Q(x) and the P(x) to make sure that when an airplane(Q(x)) wants to travel to an airport(P(x)) it is not already occupied.
2. Idea: A constraint that makes sure that Pilot B can only travel to x amount of airport. Where P(x) represents all the airports again. B(x) is a function that makes sure a pilot has travelled to an airport. B(x) can not exist for all the x values, so we use an ∃x to make sure that there are a select amount of airports Pilot B can travel to.

We would like help reaching meaningful conclusions.

Useful Notation

Feel free to copy/paste the symbols here and remove this section before submitting.

$\wedge \quad \vee \quad \neg \quad \rightarrow \quad \forall \quad \exists$