Queen's UNIVERSITY

GRP_10: Flight Booking

Tommy Janna (18taj2)

Dumitru Cernelev (18pdc1)

Marc Brasoveanu (18mtb5)

Nathanael Yao (18ny13)

Course Modelling Project

CISC/CMPE 204

Logic for Computing Science

December 10, 2020

Abstract

Our model aims to represent a flight booking system for an airline on a given day. There are a variable amount of airports, we chose four for all the examples used in this document, and a corresponding amount of pilots who fly to the airports. The booking system works on the basis of demand for flights at specific airports needing to be fulfilled each day. In each generation of the model, the demand at each airport is a randomly generated number of flights, in this document we selected a range from 0 to 4. The number of pilots for a specific generation of the model is the least number of pilots needed to fulfill the demand of each airport. Pilots begin their day at different airports in order of the highest demand. No two pilots can be at the same airport at the same time at any point in the day. This means that all pilots begin their day at unique airport and then through a series of flights throughout the day must end the day at unique 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. A pilot must move between airports at each timestep. No two pilots can be at the same airport at the same timestep. For our examples in this document the number of timesteps was set to 5 meaning that each pilot must make flights in a day.

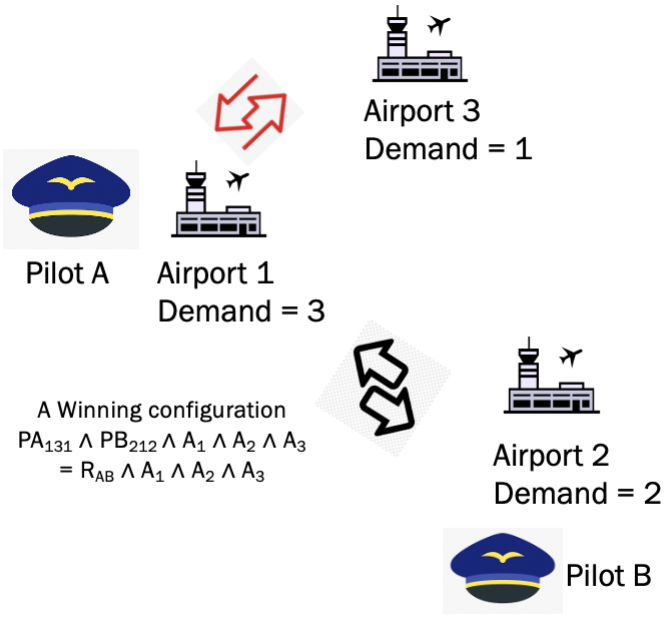


Image above shows a satisfiable scenario with two pilots, three airports for two timesteps.

Propositions

PA_{ijklm} is true when pilot A has travelled to airports i then j then k then l then m (4 flights)

PB_{abceb} is true when pilot B has travelled to airport a then b then c then e then b (4 flights)

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

A_1 is a true when the demand for airport 1 has been satisfied

In our implementation the propositions are the positions of the pilots (the airport they are at) at a given timestep.

Constraints

Our constraints are very dynamic in nature, because of the extensibility in terms of the number of airports and timesteps and the variability in terms of the demand that is randomly generated every time the program is run. This is why there is several possible versions of each constraint that we have implemented which are all equally likely to occur.

The simplest constraint we have is quite generic, however. It makes sure that a pilot cannot remain at the same airport between adjacent timesteps. That is, each pilot must make a flight on every timestep. The constraint is written as

$$\neg P_n \vee \neg P_{n+1}$$

where n is the timestep. Since the constriant is created in a loop, each of these variables are iterated through their entire domain. In other words, this constraint says that at least one of pilots location will be false between two adjacent timesteps (timestep and timestep + 1).

Next, we have a constraint that determines the starting locations of the pilots. This is a quite complex function, because if the array of demand has more equal values than the number of pilots, there will be a greater number of airports to assign than the number of pilots. So we must calculate all the different permuations of assigning n pilots to k airports. The number is modeled by this equation

$$\binom{n+r-1}{r-1}$$

We created a recursive function to determine all the possible assignments of pilots onto airports, see permute_pilots(). The constraint, in general, adds a constraint for each possible assignment, and adds disjunction between all of them. For example, if the demand for 3 airports is [3, 2, 2] and we have two

pilots, the constraint will be modeled by

$$(PA_0 \wedge PB_1) \vee (PA_0 \wedge PB_2) \vee (PA_1 \wedge PB_0) \vee (PA_2 \wedge PB_0)$$

where A and B represent two pilots and the numbers 0, 1 or 2 represent the airport they should start at. We accomplish this by generating the constraint inside of a string, such that we can add disjunctions between iterations of a loop. When we want to add the constraint, we use the python eval() function to translate the string into functional python code in
the form of the nnf type that can be interpreted by the add_constraint() function.

Another constraint we created is that a pilot can only exist on a single airport during a single timestep. We used a string here to generate the dynamic constraint again. It loops through every single pilot, timestep, and airport, and say that this specific pilot must be at one airport and no other. At the end of each airport loop, we add a disjunction symbol to the string, because the pilot can be at any one of these airports, but only one! For example, if we had only one pilot, and two airports (for the sake of simplicity), the constraint would be modelled by

$$(P_0 \wedge \neg P_1) \vee (\neg P_0 \wedge P_1)$$

where P_n represents a pilot is at airport n.

Additionally, we created a constraint that does not allow two pilots to exist at the same airport during the same timestep. We accomplished this again using a string to generate the dynamic constraint for any number of pilots/airports/timesteps. Inside each loop, we add the constraint that a pilot must exist at the given iteration airport, and that for all other pilots, they cannot exist at that given iteration airport. Outside the loop, we add a disjunction to allow any one of these possible scenarios to be true. For example, if there are two pilots and two airports, during a single timestep, this constraint can be modelled by

$$(PA_0 \wedge \neg PB_0) \vee (PA_1 \wedge \neg PB_1) \vee (\neg PA_0 \wedge PB_0) \vee (\neg PA_1 \wedge PB_1)$$

where PA_n represents the pilot A is at airport n.

Finally, we created the constraint that a pilot should only fly to an airport that has demand. As always, we opened a third-degree nested loop, and checked if the demand at the given airport is 0. If so, we also want to make sure that every other airport doesn't have a demand of 0. If every airport has a demand of 0, then the pilots should be able to fly wherever they like. After making sure that we found a demand of 0, and that not all other demands in the demand list were also 0, we are able to add the constraint that the given pilot in this iteration of the loop and at the given timestep, cannot exist at that airport! This forces pilots to fly to airports that have demand to be met! For example, if at a given timestep, airport 2 has a demand of 0, and there are three pilots, the constraint will be modelled by

$$\neg PA_2 \wedge \neg PB_2 \wedge \neg PC_2$$

Model Exploration

Our model went through several iterations, getting more and more complex. In our initial draft we had a hard coded number of pilots and airports with no demand for flights at specific airports. Our initial constraints were also hard coded for a set amount of pilots, pilot A and pilot B. For example to show that no two pilots, A and B can be on the same airport at the same time we had the following constraint: $\neg P_A \vee \neg P_B$.
The code to reflect this was, which has since been refactored:

```
### 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])
```

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. 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.

In out initial draft as outlined above we stated that all the pilots start at the same airport, which has since changed to no pilots ever being able to stay at the same airport at the same time. The previous code was:

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

Initially we had separate hard coded functions for each pilot to not be able to remain at the same airport for two time adjacent timesteps:

```
### 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])
```

This has since been changed to one generic function like many of the our previous hard coded functions:

```
### Each pilot cannot remain at the same airport in two adjacent timesteps.
for pilot in range(Pilot.count):
    for timestep in range(N_TIMESTEPS - 1):
        for airport in range(N_AIRPORTS):
            # Pilot cannot be at the same location at an incremented timestep.
            E.add_constraint(~pilots[pilot].location[timestep][airport] | ~pilots[pilot].location[timestep + 1][airport])
```

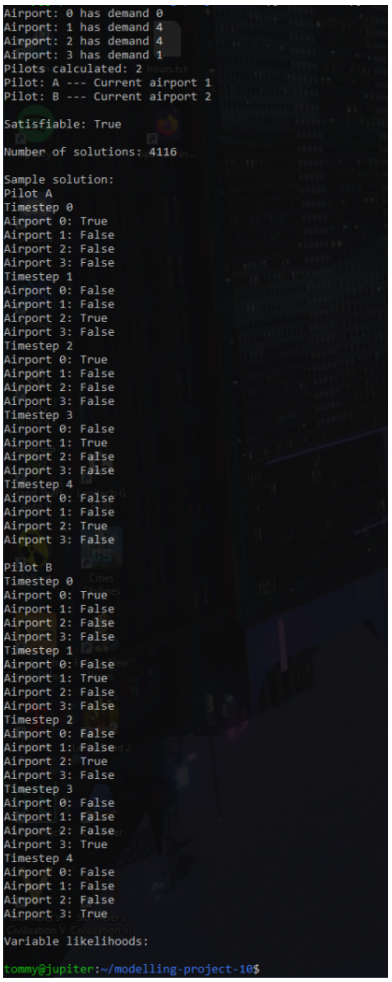
We tried testing our initial model by giving different numbers of airports, and time steps to check the number of solutions for each case and the order in which the pilots travelled.

When increasing the number of flights a pilot take per day, we discovered that this reduced the number of models that satisfy our constraints. We think this is because increasing the number of flights a pilot takes reduces the number of possible combinations. This is a sample output of 2 pilots and 3 airports.

```
Satisfiable: True

Number of solutions: 6

Sample solution:
Pilot A
Timestep 0
Airport 0: True
Airport 1: False
Airport 2: False
Timestep 1
Airport 0: False
Airport 1: True
Airport 2: False
Timestep 2
Airport 0: False
Airport 1: False
Airport 2: True
Timestep 3
Airport 0: False
Airport 1: True
Airport 2: False

Pilot B
Timestep 0
Airport 0: True
Airport 1: False
Airport 2: False
Timestep 1
Airport 0: False
Airport 1: False
Airport 2: True
Timestep 2
Airport 0: True
Airport 1: False
Airport 2: False
Timestep 3
Airport 0: True
Airport 1: False
Airport 2: False

Variable likelihoods:
```

True means that pilot A is currently at that airport and false means it is not there. We ended up changing the style of representation of a pilot path to a arrow diagram as will be seen below.

Between our initial draft and out final we wrote a function that gives us randomly generated demand for each airport and added pilots to fill this demand. We tried to explore how the model changes when we make each pilot start at specific airports. This model output can be found below.



In our final submission we fully added demand as suggested by Professor Muise. Now the model is centered around a set number of flights needing to be fulfilled at each airport on a given day. Variables like pilots now revolve around the demand since the number of pilots generated is the lowest number possible to fulfill all the demand of the airports given a set amount of timesteps in a day. Everything is fully variable now and there are no hard coded constraint or propositions, everything is generated at run time. This is what is unique with out assignment. Our constraints change based on the

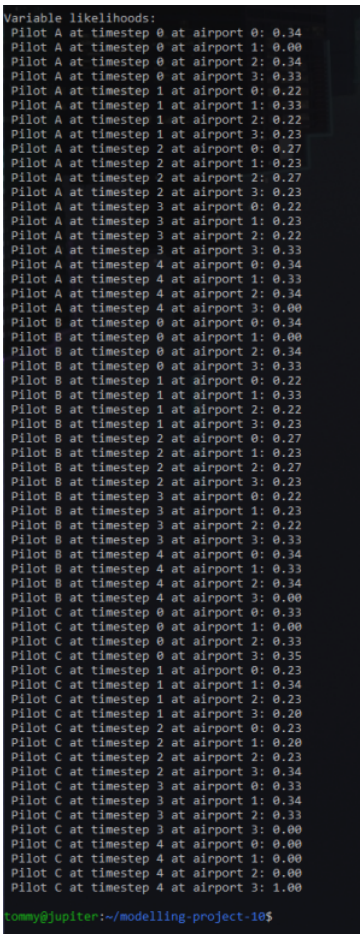
values set for certain variables like number of timesteps and demand range. Our final output looks like this for a 2 pilot model where the pilot path is denoted with an arrow diagram:

```
tommy@jupiter:~/modelling-project-10$ python3 run.py
Randomly generated demand for 4 airports...
Airport 0 has a demand of 3
Airport 1 has a demand of 2
Airport 2 has a demand of 2
Airport 3 has a demand of 0

Pilots calculated: 2
Example starting location of airports (may not be same as what T.solve() shows, but valid nonetheless)...
Pilot: A --- Starting airport 0
Pilot: B --- Starting airport 1

Satisfiable: True

Number of solutions: 256

Sample solution:
Pilot A
Airport 0 --> Airport 1 --> Airport 0 --> Airport 2 --> Airport 1

Pilot B
Airport 2 --> Airport 0 --> Airport 2 --> Airport 1 --> Airport 3
```

For a 3 pilot model the output would look like this:

```
Randomly generated demand for 4 airports...
Airport 0 has a demand of 2
Airport 1 has a demand of 2
Airport 2 has a demand of 4
Airport 3 has a demand of 4

Pilots calculated: 3
Example starting location of airports (may not be same as what T.solve() shows, but valid nonetheless)...
Pilot: A --- Starting airport 3
Pilot: B --- Starting airport 2
Pilot: C --- Starting airport 1

Satisfiable: True

Number of solutions: 449082

Sample solution:
Pilot A
Airport 2 --> Airport 0 --> Airport 3 --> Airport 1 --> Airport 0

Pilot B
Airport 3 --> Airport 1 --> Airport 0 --> Airport 1 --> Airport 2

Pilot C
Airport 0 --> Airport 2 --> Airport 1 --> Airport 0 --> Airport 3
```

In the code we have the variable likelihoods commented by default but this is a summary of the probability of a specific pilot at an airport at a timestep.



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 airport added (Airports 4, 5 ... n)

This sequent proves that if we are required to open a new airport, there exists at least one pilot that still has another flight to make after all the airports have been visited. Therefore this pilot will be available to fly to the new airport.
Predicates:
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.
D(x) is true if there is still demand at a given airport
Constraints: $\forall x.V(x) \rightarrow \exists x.(A(x) \vee B(x))$

This constraint shows that if we are required to open a new airport, there exists at least one pilot that still has another flight to make after all the airports have been visited.

$\exists x.D(x) \rightarrow \forall x.\neg((A(x) \wedge B(x)))$
This constraint shows that if there is still demand at an airport, it is not possible that all the airports have been travelled to while the pilots still have flights to make.

If we were to rewrite some of our constraints from our model using predicate logic they would look like this:

$\forall x.y.(A_1(x) \wedge \neg A1(y)) \vee (\neg A_1(x) \wedge A1(y))$

Where $A_1(x)$ is true when pilot x is at airport 1 at a given timestep. Where $A_1(y)$ is true when pilot y is at airport 1 at a given timestep.

This constraint shows that we cannot have two pilots at a specific airport at the same time.
$\forall x.y.(A(x,y) \rightarrow \neg A(x,y+1))$
Where $A(x,y)$ is true when a pilot x is at an airport at a given timestep, y.
Where x is the given pilot
Where y is the timestep

Jape Proofs Explanation

Proof 1: An airport is available, and no pilots are available imply that we need more pilots. However, in our premise we say we don't need more pilots. It is not true that airports are available and there are no pilots. So, we have more pilots available.
A = Airport Available
P = Pilots
Q = need more pilots

1: **(A∧¬P)→Q, ¬Q** premises
2: **¬(A∧¬P)** → MT 1.1,1.2

Proof 2: When a flight is scheduled. Demand is not satisfied implies that a flight is scheduled to satisfy demand, which implies that flight schedule implies a flight will occur. Therefore, A flight will occur to satisfy demand.
P = Flight scheduled
Q = Demand satisfied
S = Flight occurs

1: **P, (¬Q→P)→(P→S)** premises
2: **¬Q** assumption
3: **P** hyp 1.1
4: **¬Q→P** → intro 2-3
5: **P→S** → elim 1.2,4
6: **S** → elim 5,1.1

Proof 3: When Pilot A is free and there is still an airport that needs to be flied to, or pilot B is free and there is still an airport to be flied to. An airport that needs to be flied to implies a flight pilot A or pilot B. This means that if there is still an airport that needs to be flied to, one of the pilots must be free to fly to it.
A = Pilot A is free
B = Pilot B is free
P = There is still an airport that needs to be flied too