Algorithms for Computational Logic

Project2 Report - Group 19

José António Ribeiro da Silva Lopes
ist1103938

The Flying Tourist Problem

Imagine that a traveler wants to plan a vacation trip where he already knows the minimum and maximum number of nights he's going to spend in each city and that he will only travel by plane. The order of the flights is not predefined, and the same city in which the trip starts is the same in which it ends.

The objective of this project is to develop a tool using SMT that given a list of flights computes the most efficient travel plan: the one with the lowest cost that visits all the cities inside the specified interval of nights.

How to run the project locally

First, make sure to have `python` and `pip` installed. On Ubuntu based linux distributions this can be done by running the following commands:

```
sudo apt-get install python3
sudo apt-get install pip
```

Then, to install the `z3-solver` toolkit run the following command:

```
pip3 install z3-solver
```

And finally to run the program:

```
python3 project2.py < <input_file> > <output_file>
```

If `output_file` is not specified, the program prints the result to `stdout`.

The format of the output is in compliance with the one represented in the project statement. Which means:

- One line with an integer defining the overall final cost;
- A sequence of n lines with the chronological sequence of flights defining the trip. Each line contains the date of the flight, the city of departure, the city of arrival, the departure time and the cost of the flight, separated by one white space. All lines terminate with an end of line character.

For all explanations below assume that n is the number of cities to be visited, m the number of flights, $\mathcal{F}$ the set of all the flights, *base* the beginning and end city, K_{min} the minimum possible number of nights spent traveling and K_{max} the maximum possible number of nights spent traveling.

Variables used

The variables used in this project and in Project1 were the same. All of them are Boolean variables.

- $\forall i \in \{1..n\} : c_i$ is true if the traveler has left city i.

- $\forall i \in \{1..n\} : c'_i$ is true if the traveler has arrived at city i.
- $\forall i \in \{1..n\} : f_i$ is true if the flight i has been taken.

Encodings

1. $\forall i \in \{1..n\} : c_i \land c' i$, meaning that the traveler has to have left every city and arrived at every city.
2. Assume that $i = 1$ is the index of the *base* city. Then, $\forall i \in \{2..n\} : (\neg c_1 \Rightarrow \neg c_i)$, meaning that if the traveler hasn't left the *base* city then he can't have left any other city. This forces the trip to start at *base* city. Subsequently, $\forall i \in \{2..n\} : (c'_1 \Rightarrow c'_i)$, meaning that if the traveler has arrived at the *base* city then he also has to have arrived at every other city. This forces the trip to end at *base* city.
3. $\forall i \in \{1..m\} : \forall j \in \{1..n\}$: If $\mathcal{F}_i$.departure $= j$ then $\left(f_i \Rightarrow c_j\right)$, meaning that if the traveler takes a flight from city j, he has certainly left that city.
4. $\forall i \in \{1..m\} : \forall j \in \{1..n\}$: If $\mathcal{F}_i$.arrival $= j$ then $\left(f_i \Rightarrow c'_j\right)$, meaning that if the traveler takes a flight to city j, he has certainly arrived at that city.
5. Assume that *first_day* is the day of the first flight. Then, $\forall i \in \{1..m\}$: if $\mathcal{F}_i$.date $=$ *first_day* and $\mathcal{F}_i$.departure $\neq$ *base* : $(\neg f_i)$, meaning the traveler can't be leaving any city other that *base* on the first day.
6. Assume that *first_day* is the day of the first flight. Then, $\forall i \in \{1..m\}$: if $\mathcal{F}_i$.date $-$ *first_day* $< K_{\min}$ and $\mathcal{F}_i$.arrival $=$ *base* : $(\neg f_i)$, meaning the traveler can't be going back to *base* if $K_{\min}$ nights haven't passed.
7. Assume that *last_day* is the day of the last flight. Then, $\forall i \in \{1..m\}$: if $\mathcal{F}_i$.date $=$ *last_day* and $\mathcal{F}_i$.departure $=$ *base* : $(\neg f_i)$, meaning the traveler can't be leaving *base* on the last day.
8. $\forall i \in \{1..m\}, \forall j \in \{i + 1..m\}$: If $\mathcal{F}_i$.depart $= \mathcal{F}_j$.arrival and $\mathcal{F}_i$.depart $\neq$ *base* : $\left(f_i \Rightarrow \neg f_j\right)$, meaning that if the traveler leaves a city, no more flights to that city can be taken, except for *base.*
9. $\forall i \in \{1..m\}, \forall j \in \{i + 1..m\}$: If $\mathcal{F}_i$.arrival $= \mathcal{F}_j$.depart and $\mathcal{F}_j$.date $- \mathcal{F}_i$.date $\neq$ $\mathcal{F}_i$.arrival.city.nights : $\left(f_i \Rightarrow \neg f_j\right)$, meaning that if the traveler takes a flight to a city c then if a flight leaving c doesn't do so in in the interval of $\left[k_{\min_c}, k_{\max_c}\right]$ nights, it cannot be taken.

Some of this encodings are not necessary to ensure the correctness of the algorithm, but are there to help the SMT solver reduce searching space, for example the one's that eliminate the possibility of leaving the *base* city in the last day or arriving at the *base* city on the first day.

Algorithm and configurations

The SMT solver used was `Optimize` provided by the `z3-solver`, with the objective of minimizing the sum of the prices of all the flights that were taken.

Experimental evaluation

Another possible SMT encoding was tried, where a new variable was added: $\forall i \in \{1..n\} : n_i$ is the number of nights spent at city i and the encoding nº9 was replaced by the following one:

- $\forall i \in \{1..m\}, \forall j \in \{i + 1..m\}$: If $\mathcal{F}_i$.arrival $= \mathcal{F}_j$.depart : $\left(\left(f_i \land f_j\right) \Rightarrow n_{\text{city}\mathcal{F}\text{depart}} = \mathcal{F}_j.\text{date} - \mathcal{F}_i.\text{date}\right)$, meaning that if a flight to a city is taken and a flight from that city is taken some days later, that the number of nights spent in that city must be equal to the difference of the dates of said flights.

Both this new SMT encoding and the one explained in the previous sections got timed out in test $t16$ of the public instances.

Excluding that test, the old SMT encoding took **12 minutes and 59 seconds** to run all the public instances and the new one took **10 minutes and 49 seconds**. Even though the old one is slower to run the public instances, I decided to stick with it since it was passing all the private tests on the course's GitLab, while the new one got a Time Limit Error on test 15. Either way, the code with the new encoding can be analyzed in the file `project2_new.py`.