AIRLINE PLANNING ALGORITHM

Oleguer Canal - Fernando Garcia - Federico Taschin - Catherine Weldon

KTH - Royal Institute of Technology

Integer Programming

For the *Integer Programming* approach, the problem was solved using *AMPL*, a modeling tool specifically designed for optimization problems. Because the problem's constraints involved multiple "if-statements" and the maximization called for multiple variables, the optimization problem was non-linear.

1 What is *Integer Programming*

Integer programming is a mathematical optimization method in which all variables are restricted to be integers.

2 Simplified Implementation

Parameters:

E_n Destination for each passenger n

Variables:

$x_{ijt} = \begin{cases} 1 & \text{If flight travels from Origin i to Destination j at point t} \\ 0 & \text{Otherwise} \end{cases}$

$m^n_{ijt} = \begin{cases} 1 & \text{if passenger n travels from Origin i to Destination j at point t} \\ 0 & \text{Otherwise} \end{cases}$

Objective $\rightarrow$ MAXIMIZE Profit (people moved - flight cost)

$$\sum_{n=1}^{N} x_{ijt} m^n{}_{ijt} E^n{}_j - \sum_{ijt} x_{ijt}$$

3 Conclusions

- **Problem 0** was solvable under certain conditions with Integer Programming.
- Default *AMPL* solver, Minos does not guarantee a global optimum for non-linear problems.
- Many of the formulations with stricter time-limits or additional passengers resulted in non-optimal routes.
- *AMPL* computing limits are also a problem, as the solver would return a solution before completing all necessary iterations if the scale was too large.
- **Problem 1** and **Problem 2** were not implemented with Integer Programming methods.

PDDL

This is a planning problem, therefore it can easily be expressed with PDDL and solved with a PDDL solver.

1 What is *PDDL*

Programming Domain Definition Language (*PDDL*) is a standardized planning language for AI systems.

2 Simplified Implementation

Passengers are grouped by final destination to reduce branching factor. A plane can either board all or none of the passengers in a group. The used actions are the following:

- **Action BOARD**(plane, group, city):
  - **Preconditions**: The group and plane must be in the same city. Deadline must not be reached.
  - **Effects:** Group is boarded onto the plane and is no longer in the city. Onboard passengers' counter and plane stopwatch are set accordingly.
- **Action FLY**(plane, from, to):
  - **Preconditions**: Plane did not reach deadline and is in city `from`.
  - **Effects**: Move the plane in city `to`, increase plane stopwatch by the time between `from` and `to`.
- **Action UNBOARD**(plane, group, city):
  - **Preconditions**: Group is in plane and plane is in city.
  - **Effects**: Move group to the city, set empty seats in the plane, set group stopwatch accordingly.

3 Conclusions

- **Problem 0** was solvable with either equal or more optimal results than the other approaches for each problem variation. However, the number of evaluated states quickly explodes as the problem size increases, resulting in very long computation times and/or out of memory errors.
- **Problem 1** was solvable for small problem instances. The PDDL solver can handle a great number of passengers when grouping is possible, but fails in all other cases.
- A great limitation was given by the scarcity of properly working open source PDDL solvers. Although many solvers (Metric-FF, FastDownward, ENHSP) were tried, only one (ENHSP) was able to handle the problem, and not all of its search/heuristic options returned a solution.

Reinforcement Learning

An *Active Reinforcement Learning* solution is presented in order to model stochasticity, added constraints, and solve problems within a larger state space and bigger branching factor. In particular, the algorithm used is *Q-Learning* with *epsilon-greedy* exploration.

1 What is *Reinforcement Learning*

Reinforcement Learning is a machine learning area concerned with how software agents should take actions in a given environment such that some notion of a cumulative reward is maximized.

2 Simplified Implementation

- **State**:
  - Cities considered and distances between them
  - List of people with current and desired cities
  - List of landed planes and their locations
  - List of ongoing flights
- **Action**:
  - List of all possible combinations of flights given current state
- **Reward**:
  - Moving people to their final location
  - Moving planes towards cities with more people
  - Minimizing total time
  - Minimizing airtime

Note: Stochasticity is added in the form of probability of people missing a flight once the action has been taken

3 Conclusions

- **Branching factor:** Better manages larger state spaces compared to other methods but it is still a problem.
- **Meta-parameters:** Highly susceptible to: episode number, learning rate, discount factor, epsilon, reward weights ...
- **Quality Table Limitations:** Simple state-action table can be improved with a deep network instead.
- **Algorithm generalization:** Current defined state space could easily be tested with different algorithms such as simulated annealing[?] or genetics algorithm [?].