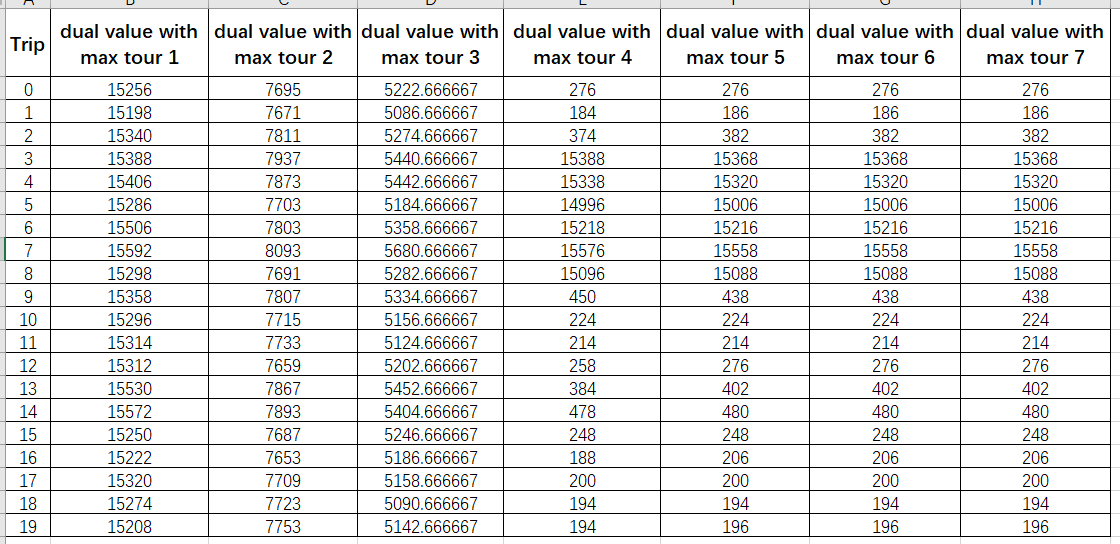
**3.4 Branch and Price**

For a given twenty trips, restricting the max tour, the max numbers that each schedule can do, from 1 to 7. Calculating the , the dual value of trip t with max tour equals to mt and made the table below.



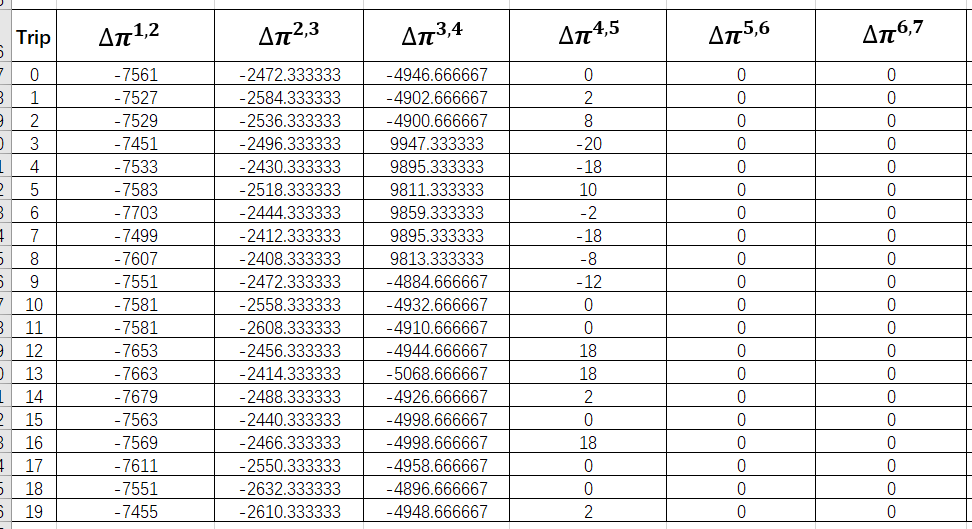
**Table(3.4-1)**

At the 3rd column of Table(3.4-1), when the max tour equals to 2, the algorithm generates a schedule, (4, (0, 17))(Basically means a schedule starts from depot-4 with have trip-0,trip-17)with the price 15404. We can see in 3rd column, the dual value of trip 0 is 7695 and the dual value of trip 17 is 7709 which means the price 15404 = 7695() +7709().

Interestingly, =++–1 and =+++1 .

So, we can deduce the dual values in the algorithm explaining the cost that’s a schedule needs to pay if it wants to contain a certainly trip. The cost can be made by the cost of the trip itself and the cost of driving to the start of that trip, and, it also shares a nearly equivalent percent of the BusCost with other trips in the schedule when all the length of the schedule is equivalent.

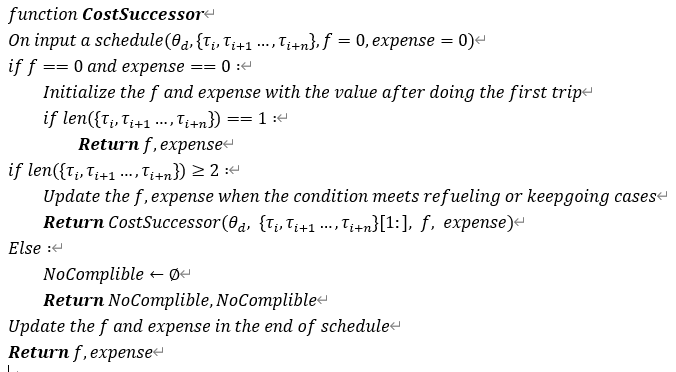
However, in the case max tour is greater or equal than 4, we can see that from each column, trip 4,5,6,7,8 has the biggest dual value and in the calculated schedule each of these trips are not contained in a same schedule (Otherwise it would not be possible).



**Table(3.4-2)**

From the table abrove we can see that the gap between each step’s dual value is fluctuating and decreasing to zero when nothing is changed. Which give us an idea about if we pick the right trip and package it into a schedule, the total dual price would go down. More detailed to say, if we calculate the dual price correctly with the candidate trip, we can decide whether it can be contained in some king of schedule. Which would be different from the ordinary column generation and save much time. Because the normal generation process would only contain the feasible column independently instead of taking the whole processing into consideration. We eliminated the trip cost from the cost calculating algorithm because no matter what the schedule is, these costs would always be costed.

We make a general function for calculating the current fuel and expense with a particularly schedule.



**Figure(3.4-1)**

To start, we follow general approach of Danztig-Wolfe Decomposition:

1. Generate initial set of Variables

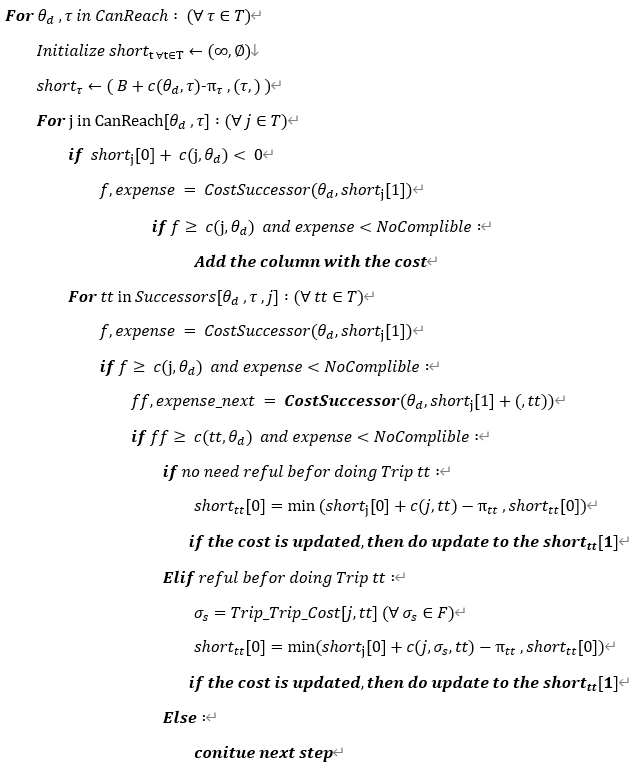
2. Solve the resulting Restricted Master Problem.

3. Use the dual variables to find new Master Problem variables which will improve the objective.

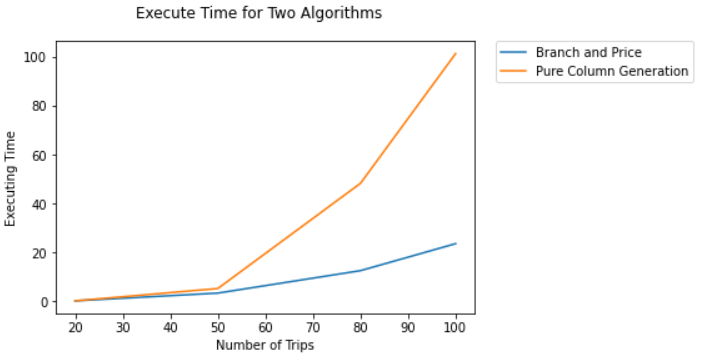
Denoted as the dual value of trip t. The dual value of is denoted as :

Dual()=

We keep updating the dual value of , and do the column generation when we find a value that is less than the current value for .

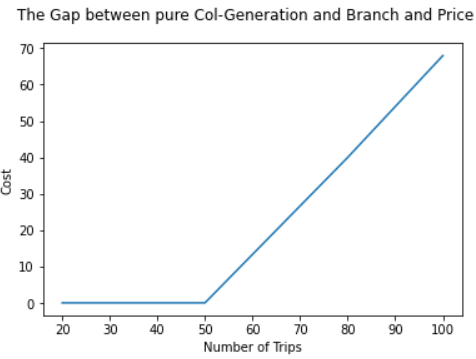


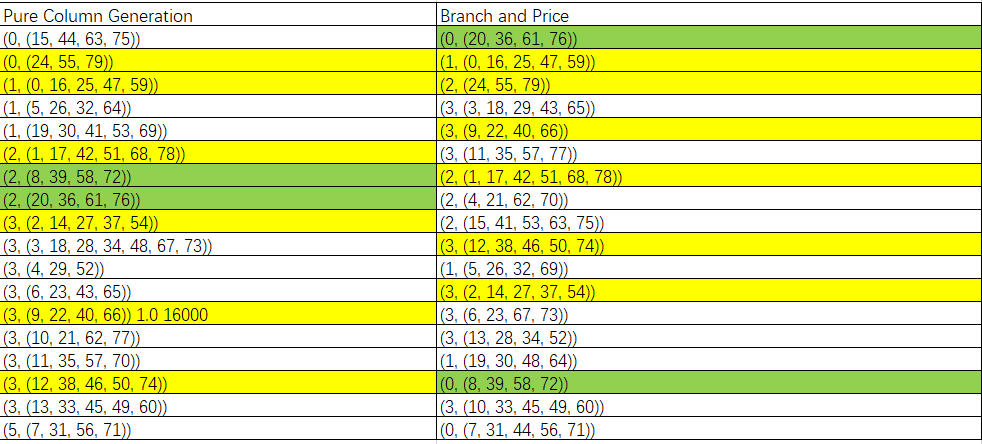
**Figure(3.4-2)**

 **Figure(3.4-3)**

We can see a huge improvement by implementing the Branch and Price algorithm. What’s for sure is that the algorithm can certainly be revised and saving a huge amount time because there is a lot of code checking if the current station is feasible to continue and many of them would be redundant. Moreover, the algorithm **CostSuccess** takes a complete trip chain and compute the current condition from the beginning is repetitive. Whereas, it seems there is no better way saving the fuel condition or calculating it fleetly.

Sadly, the algorithm itself is not complete correct. We deduced the outcome could give the number of correct buses so there is a little bug. It could be the reduced cost in refueling condition or some code just restrict the generation of feasible trip chain.

**Figure(3.4-3)**



**Figure(3.4-4)**

The figure (3.4-4) lists the two algorithms’ outcome for 80 trips and without any constrain but only going back to the starter depot, meeting the time and fuel constrain.

The yellow line is just entirely two same trips and the green line is the same trip with different starting depot. We can just focus on the first green column on the right and see why the schedule have not worked from deport 2. Interestingly, you can also see the sequence, for pure column generation, the algorithm first fixed the depot and do the circulation for trips. For Branch and Price, the sequence may tell us what’s the first trip chain that is generated in the algorithm. Anyway, we would keep fixing what’s going on and emailed you as soon as we solved.

When talking about the Branch and Price, the paper said that when a non-integer solution coming, as in my algorithm I usually got 0.5 for two trip chains contained by a same trip. In paper, the author let it branches into two cases, one trip chain with pair must stay together, the other trip must not the pair can stay together. Then, at the end of the Branch and Price part, the author also talks that he used a list to save these branch point so that for the following iteration it could come to a fast react and just exclude the pair for being excluded by the former branches.