INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Department of Industrial and Management Engineering

IME639A - ANALYTICS IN TRANSPORT AND TELECOM

Course Instructor - Dr. Faiz Hamid

Bus Driver Scheduling Problem

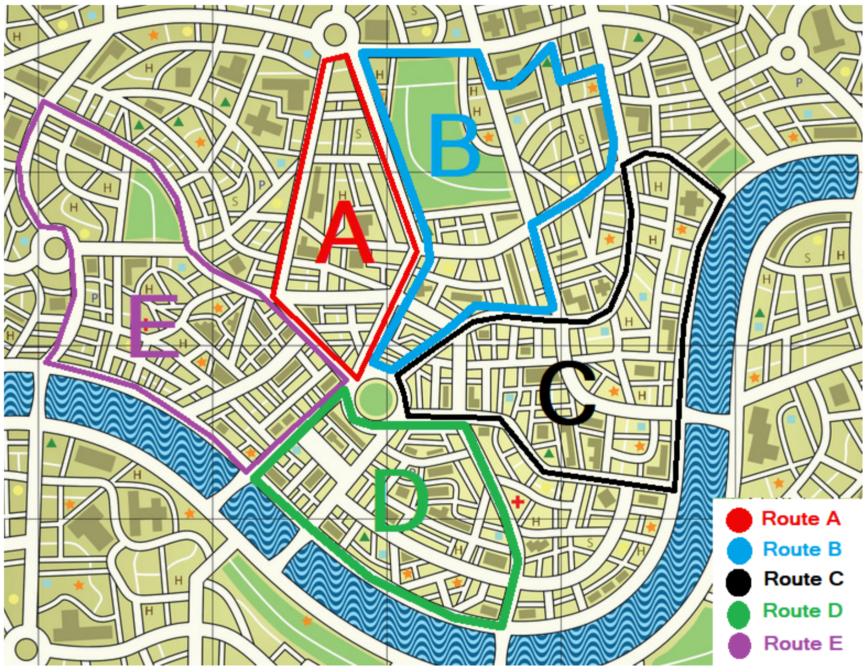
200782	Richa Prakash Sachan
201093	Varun Pant
190533	Narender Kumar Dhaka
190821	Shreya Bhiwaniya

Table of Contents

The Problem

The Bus-Driver Scheduling Problem is a well known problem in operations research and theoretical computer science. There are many formulations of the problem depending on the various constraints and the optimization objective being achieved. The problem that we have solved during the IP optimization goes as follows.

- There are five bus routes, namely A, B, C, D and E; each of which is a closed loop in itself.
- Each route takes an hour to complete.
- A single bus runs on each route starting from 5AM in the morning until 11PM in the night.
- Moreover, all the routes start and end at a common point every hour on the hour, making it convenient for the drivers to take their meal breaks and shift changes at that location.



Since our bus service runs from 5 AM until 11 PM, and each route takes an hour to complete, we'll have 18 full trips for each route (90 total trips). Our goal is to create a roster of duties for the bus drivers, where **each duty is a sequence of trips that a driver is required to operate.**
It is essential to ensure that every trip is included in precisely one driver's duty.
There are some guidelines from payroll that we need to consider while creating the duties for the bus drivers:

1. **Each duty over four hours should allow the driver to have a break.**

2. **The drivers will receive payment for each hour they work, excluding their break time.**
3. **The drivers will receive guaranteed pay for eight hours of work, irrespective of the actual number of hours they work.**
4. **If a driver works for more than eight hours, they will receive overtime pay, which is 1.5 times their regular pay rate.**

Optimization Task - Our task is to create a roster of duties which **minimizes the driving costs**.

IP Formulation

Our approach of solving the above problem is modeling it as a *weighted set partitioning problem*.
Set Partitioning problem - We have a set of different items and then we make sets containing different combinations of items out of them. The solution to the set partitioning problem is to find the smallest number of sets that contain all the items (points), without any overlap between sets (hence a "partition"). Every point must be in one and exactly one set.
Weighted Set Partitioning problem - When a cost is assigned to each set, we have a weighted set partitioning problem where the goal is to minimize the sum of the values of the sets we choose instead of minimizing the number of sets. If we set all weights to one, the problem becomes the original set partitioning problem.
Both of the above problems are NP-complete, however we can still use integer programming to solve the given problem optimally.

- In our specific problem, the items refer to trips a bus takes along a route at a particular start time, and the sets are potential driver duties, meaning each duty is a set of trips that a driver could potentially undertake.
- Each of these sets must satisfy the constraint that over 4 hours of duty, the driver should get a break.
- To these sets, we assign a weight using the driver pay rules defined earlier, such as overtime, guarantee pay, and others.
  Once we solve the weighted set partitioning problem, we'll obtain the set of duties that covers all the trips exactly once, while minimizing the sum of individual drivers' costs, which is precisely our desired outcome.

To implement this solution, we will assign each potential duty as a variable (an x_i) in our integer program. We will constrain all x_i values to be either zero or one. The concept behind this constraint is that a specific x_i that will have a value of one in the solution, if the solver determines that using the corresponding duty as the part of the optimal solution. If the duty is not included in the optimal solution, then the value of the corresponding x_i variable will be zero.

The coefficient p_i of each x_i variable represents the cost of the duty in terms of driver pay.

$$\textbf{Minimize objective function}(x_1, x_2, x_3,....,x_n) = p_1x_1 + p_2x_2 +...... + p_nx_n \text{; each } x_i=\{0 \text{ or } 1\}$$

Since every trip needs to be part of exactly one duty, we can add a constraint for each trip. These constraints ensure that the sum of x_i variables corresponding to the duties that the trip is a part of must be equal to 1. This constraint guarantees that each trip is assigned to exactly one duty in the solution.

For each trip t_k let x_j be the binary variables corresponding to the duties containing t_k then $\mathbf{\Sigma x_j = 1}$

Generating the possible duties -

- **This is an exponentially complex problem**. For instance in case of our problem we have 90 trips, the worst case possible number of duties is = 2^{90}. This is enormously large!! Therefore, **we will randomly create only a certain number of duties from the set of trips**. For this we are using a function that randomly generates duties while ensuring that no duty includes multiple trips that occur simultaneously.
- Since our objective is to create a set of duties that includes all the trips. However, as we are randomly generating duties, we cannot guarantee this. There is a possibility that all randomly generated duties may not include all the trips on a particular route. To address this issue, we can generate one duty for each trip, which only includes that specific trip. This ensures that each trip is included at least once in the set of duties.

Implementation and solution of IP formulation in Python

To solve this problem in Python we have employed the **PuLP solver**. Due permission in this regard was taken from the course instructor. The code file ***Group11_IME639A_BDSP.ipynb*** was created in Jupyter Notebook and has in detail explanation to the problem and appropriate comments corresponding to all the parts in the code.

- *list_all_trips* - This subroutine generates a list of all possible trips
- *generate_duties(num_duties, trips)* - This subroutine generates randomly a list of duties of cardinality = *num_duties* from the *trips* while ensuring that no duty includes

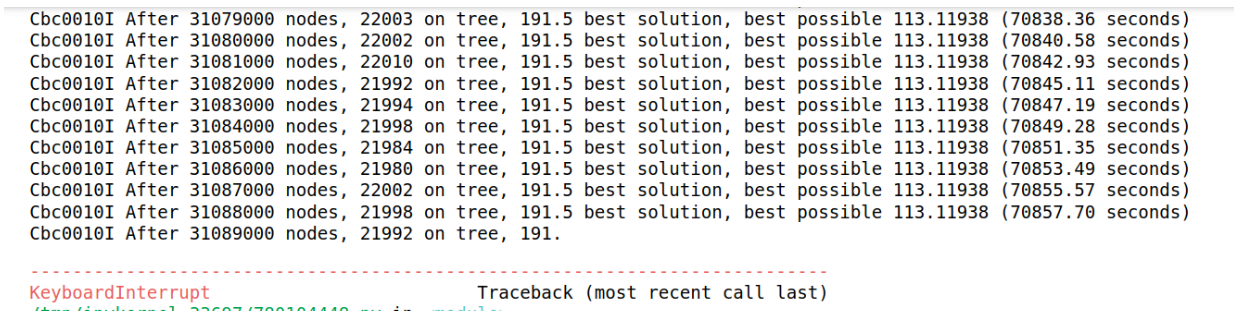
multiple trips that occur simultaneously. However this **doesn't ensure that all the trips have been included in the duties thus generated.** This needs to be ensured that our possible set of duties has all trips included at least in one of the duties because we cannot miss on any trip, each trip has to be present exactly once in the optimal solution. If it will be absent from a set of possible duties, out of which we shall be formulating our optimal solution, then how will it be present in the optimal solution?? **This also doesn't ensure the first payroll constraint, which doesn't permit us to have duties spanning more than 4 hours at a stretch.**

- *cost(duty)* - This subroutine assigns the cost to a given *duty* as per the rules 2,3 and 4 of the payroll. **We define 1 unit as a standard pay for 1 hour of work**. In this function **we also ensure that we enforce the first payroll constraint** by assigning very very high costs to duties which violate it. Since our problem is a minimization problem, they will not be counted into the optimal solution while solving.
- *solve(duties, trips)* - This subroutine solves for the set of *duties* and *trips* providing an IP optimal solution. It is in this function that we enforce that from the set of *trips* every trip is assigned to one and only one duty, and not assigned to multiple duties or left unassigned in the optimal solution.

We have solved the problem for three different sizes of the list of duties generated randomly. With increasing size of the list of duties we observe that the value of the objective function corresponding to the optimal solution thus generated is significantly improved. However the time elapsed by the solver in solving the problem also increased significantly with increase in the size of the list of duties.

Size of the list of randomly generated Duties	**Value of the objective function corresponding to the optimal solution**	**Time taken to generate the optimal solution**
5	597.5	milliseconds
100	341.0	1.92 seconds
1000	**191.5**	**>20hours**

Corresponding to the *Size of the list of randomly generated Duties* = 1000, the program didn't give the output even after 20 hours of running. The program was terminated due to the time limit on the assignment deadline, however, the value of objective function when program was terminated came up to be 191.5. The proof is also in ss below.



```
Cbc0010I After 31079000 nodes, 22003 on tree, 191.5 best solution, best possible 113.11938 (70838.36 seconds)
Cbc0010I After 31080000 nodes, 22002 on tree, 191.5 best solution, best possible 113.11938 (70840.58 seconds)
Cbc0010I After 31081000 nodes, 22010 on tree, 191.5 best solution, best possible 113.11938 (70842.93 seconds)
Cbc0010I After 31082000 nodes, 21992 on tree, 191.5 best solution, best possible 113.11938 (70845.11 seconds)
Cbc0010I After 31083000 nodes, 21994 on tree, 191.5 best solution, best possible 113.11938 (70847.19 seconds)
Cbc0010I After 31084000 nodes, 21998 on tree, 191.5 best solution, best possible 113.11938 (70849.28 seconds)
Cbc0010I After 31085000 nodes, 21984 on tree, 191.5 best solution, best possible 113.11938 (70851.35 seconds)
Cbc0010I After 31086000 nodes, 21980 on tree, 191.5 best solution, best possible 113.11938 (70853.49 seconds)
Cbc0010I After 31087000 nodes, 22002 on tree, 191.5 best solution, best possible 113.11938 (70855.57 seconds)
Cbc0010I After 31088000 nodes, 21998 on tree, 191.5 best solution, best possible 113.11938 (70857.70 seconds)
Cbc0010I After 31089000 nodes, 21992 on tree, 191.

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
```

Cbc0010I After 31088000 nodes, 21998 on tree, ***191.5 best solution****, best possible 113.11938 (****70857.70 seconds****)*

Best solution is objective function value and time elapsed is 70857 seconds, nearly equal to 20 hours.

Heuristic based algorithm for solving this problem

So, for the problem given, the heuristic based approach we figured out appears to give us the most optimal solution to the problem. This performs much much better than the branch-and-bound of the PuLP solver. **It yields an ultimate optimal solution with the value of objective function = 91**, that too in much less time(compared to the time taken by IP solver). So let's have a look at the approach.

1. **Heuristic 1** - We have to pay each driver a minimum pay equalling 8 hours of work, even if he/she covers less than 8 trips. So the **first heuristic we can think of is to assign each driver at least 8 hours of work, that is 8 trips, so that our pay doesn't get wasted**.
2. **Heuristic 2** - If a driver takes more than 8 trips, then we will have to pay the driver 1.5 times the cost of a trip, for each trip he does over his 8. So **for minimizing the total cost, it is preferred that we assign less number of drivers, more than 8 trips because doing so will cost us 50% more on each such trip**.
3. **Heuristic 3** - Since all trips are identical in terms of the time taken in them and the payrolls they would offer; **it doesn't make any difference if at the same time of the day we swap the routes of two or more drivers**. This can be explained as follows, suppose the optimal solution says that at 11AM all the 5 routes are assigned to the drivers as follows:

Driver i	Driver i+1	Driver i+2	Driver i+3	Driver i+4
(A,11)	(B,11)	(C,11)	(D,11)	(E,11)

   Then it won't make any difference in the optimal solution if we shuffle the routes on A,B,C,D,E among the 5 drivers above.

From the first and second heuristic, we can compute a lower bound on the value of objective function.

Total trips = 90

We prefer most number of drivers to have exactly 8 trips, so we can have:

90/8 = 11 drivers covering the 88 trips + 2 trips left

For the two trips, if we hire a new driver, it costs 8 units, but if we assign it to one or two of the already existing drivers, it costs 1.5*2=3 units. So we go with the second case.

The total cost we therefore get is = (11 drivers*8 units) + 3 units = **91 units**

The lower bound on objective function using first and second heuristic is 91.

Taking the **third heuristic into account** we see that optimizing the driver duties on each route individually, and doing this for all the 5 routes together, would lead us near to the optimal solution.
Each route has 18 trips, a driver can take at most 4 of them consecutively; these 18 trips can be broken down as:

18 trips = 4+1+4+4+1+4 trips

4(5AM-9AM)	1(9AM-10AM)	4(10AM-2PM)	4(2PM-6PM)	1(6PM-7PM)	4(7PM-11PM)

Using the **first heuristic** we assign at least 8 hours of duty to each driver. Therefore for each route we can have two drivers doing at least 8 hours.

4(5AM-9AM)	1(9AM-10AM)	4(10AM-2PM)	4(2PM-6PM)	1(6PM-7PM)	4(7PM-11PM)
Driver 1		Driver 1	Driver 2		Driver 2

For all the five routes this becomes as follows:

	4(5AM-9AM)	1(9AM-10AM)	4(10AM-2PM)	4(2PM-6PM)	1(6PM-7PM)	4(7PM-11PM)
A	Driver 1		Driver 1	Driver 2		Driver 2
B	Driver 3		Driver 3	Driver 4		Driver 4
C	Driver 5		Driver 5	Driver 6		Driver 6
D	Driver 7		Driver 7	Driver 8		Driver 8
E	Driver 9		Driver 9	Driver 10		Driver 10

In the assignment above, we can be confident from the first heuristic that we are making best possible use of the 10 drivers.

If we further go on to optimize just a single route, for the two trips that provide breaks to Driver 1 and Driver 2 on route A;

we can either hire a new Driver 3 as follows:

4(5AM-9AM)	1(9AM-10AM)	4(10AM-2PM)	4(2PM-6PM)	1(6PM-7PM)	4(7PM-11PM)
Driver 1	Driver 3	Driver 1	Driver 2	Driver 3	Driver 2

or assign them to the Driver 1 and Driver 2 itself as follows:

4(5AM-9AM)	1(9AM-10AM)	4(10AM-2PM)	4(2PM-6PM)	1(6PM-7PM)	4(7PM-11PM)
Driver 1	Driver 2	Driver 1	Driver 2	Driver 1	Driver 2

The first approach costs us 24 units on each path, whereas the second approach costs us 19 on each path, indicating that the latter one is more optimal. Following this approach for all routes gives us:

	4(5AM-9AM)	1(9AM-10AM)	4(10AM-2PM)	4(2PM-6PM)	1(6PM-7PM)	4(7PM-11PM)
A	Driver 1	Driver 2	Driver 1	Driver 2	Driver 1	Driver 2
B	Driver 3	Driver 4	Driver 3	Driver 4	Driver 3	Driver 4
C	Driver 5	Driver 6	Driver 5	Driver 6	Driver 5	Driver 6
D	Driver 7	Driver 8	Driver 7	Driver 8	Driver 7	Driver 8
E	Driver 9	Driver 10	Driver 9	Driver 10	Driver 9	Driver 10

For the assignment as above the value of objective function we get is 95 units. This value is just 4 units more than the lower bound of 91 and is still much much better than the value we obtained by solving the problem as an IP optimization in Python.

Can this be improved further??
Well, Yes!
Comparing our solution with the lower bound solution, there are 8 trips for which we are paying 1.5x more. If we are somehow able to optimize these 8 trips in our solution, we will be able to reach the lower bound. To do this we would need a new driver who would cover only these 8 trips. This will cost us 1 unit per trip.
We observe that in the solution we have formulated above the breaks for the drivers occur at the same time in each of the routes. If we somehow shift the breaks on 4 of the routes to occur all at different times and assign them to the new driver, also ensuring that there is a gap of 1 hour between the 4 consecutive breaks, we are done with getting the best possible solution of the problem.
A simple effort for the same on a spreadsheet shows us that it is indeed possible. Just see below.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A	D1	D2				D1	D3		D2			D3		D2	D3			
B	D4	D1	D4				D1	D5				D4			D5			
C	D6		D1	D6				D1	D7				D6		D7			
D	D8			D1	D9				D1	D8	D9				D8			
E	D10				D11	D10				D11				D10	D11			

The assignment of duties as above costs us just 91 units, or the objective function is 91, which is equal to the lower bound for the minimization problem.

Conclusion

1. We stated the Bus-Driver scheduling problem completely with all constraints.
2. We formulated it as an IP problem and attempted solving it in Python with the PuLP solver. We were able to get three optimal solutions corresponding to different sizes of duties we had and also observe the computational complexities involved with them.
3. Using the heuristic based approach not only did we get an optimal solution whose objective value is much better than the branch and bound of the IP solver in much less time, we also got the solution to be the most optimal one with the value of the objective function equaling the lower bound value.