Optimizing BART Transit Scheduling

Ian Hall and **Ian Rolls**
{iahall,iarolls}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

1 Introduction

Efficient scheduling for transit networks is essential for keeping trains on schedule, preventing train line clog-ups, and ensuring all passengers are able to access the network in a timely manner. As a network gets larger and larger, this becomes even more important, as the network needs to be able to handle more station/line permutations. We look at the problem of scheduling trains on a heavy rail line, specifically, San Francisco's BART transit system. We use a mixed integer linear programming (MILP) approach to find how many train cars should be running on each track during a given time period of a weekday.

2 Problem Formation

The San Francisco **B**ay **A**rea **R**apid **T**ransit District (**BART**) is a heavy-rail public transit system in the San Francisco Area. BART operates in five counties (San Francisco, San Mateo, Alameda, Contra Costa, and Santa Clara) with 131 miles of track and 50 stations (`bart.gov`, 2023). Although globally this would be considered a small to mid-sized transit network, BART is the second largest transit network in the United States by distance after New York (248mi of track). Additionally, BART is the 7th largest network in the US in terms of yearly ridership.

We first examine the BART transit map to formulate our linear programming. As you can see in figure 1 which is BART's official transit map, there are 6 different rail lines represented by different colors. It is important to note that many of these color lines overlap, with certain sequential stations both having four lines, like Balboa Park and Glen Park. For this reason, we chose to break up the transit map by physically separated rail lines, of which `bart.gov` provides the distinctions. Table 1 displays these distinctions and their corresponding datapoints, including the start/end stations of the track, the distance of the track (d_i), and the time it takes (in hours) for a train to complete a one-way trip on the line assuming the given average speed of 35mph (t_i).

Variable	Start Station	End Station	Distance (d_i)	Time of Trip (t_i)
ℓ_1	Barryessa	Oakland West	38.9mi	1.11hrs
ℓ_2	West Oakland	Millbrae	27mi	.77hrs
ℓ_3	Richmond	Oakland West	13.2mi	.377hrs
ℓ_4	Pittsburg Bay	Oakland West	32.3mi	.923hrs
ℓ_5	Antioch	Pittsburg Bay	9.1mi	.26hrs
ℓ_6	Oakland West	Dublin	24.7mi	.706hrs
ℓ_7	Coliseum	OAK Airport	3.2mi	.091hrs

Table 1: BART Physical Line Distinctions

By separating these lines by physical tracks, we assume that the trains on each line must carry all passengers that utilize that physical line in their route, irrespective of which color line they take. Thus, we don't need to make extra constraints for these color lines or get data for which specific color line path each individual takes to get to their destination or where individuals transfer lines.

Our overall goal is to minimize the network costs while also meeting all passenger goals. We do not have to create a specific cost function because all variable costs of the network are a function of total distance travelled by each train (energy and operation costs) and the total number of trains (servicing and conducting costs). Thus, if we minimize the total distance travelled by each train, we should in turn be minimizing the total cost of the network.

Figure 1: BART Heavy Rail Line Network Map

3 Data Description

Next, we needed ridership data to create the constraints so each train can meet the number of passengers travelling on the network. `bart.gov` provides a weighted adjacency matrix for their entire train network. Specifically, they have data on the average weekday ridership for each month from any one station i to any other station j on the map. This is essentially a 50 by 50 weighted adjacency matrix where each entry $A_{(i,j)}$ contains the number of riders who go from station i to station j. We used the March 2023 data, as it is the most recent, and ridership is more likely to increase in the warmer months as it is more pleasant to wait for a train in an outdoor station in warmer weather.

We then created a corresponding matrix B so each $A_{(i,j)}$ in the original matrix corresponds to a list of numbers in B. This list of numbers represents the different lines that a passenger would need to use to get from their starting point i to their final destination station j. These lines and their corresponding numbers 1-8 are displayed above in table 1. For example, if an average of 67 people go from Richmond to Millbrae, they would have to take ℓ_3 to West Oakland and then take ℓ_2 to Millbrae. Thus, the entry in $A_{(i,j)}$ is 67 and the corresponding entry in $B_{(i,j)}$ would be $[2, 3]$. In our modelling, we require each train line to be able to accommodate all riders that use that train line in their start $\rightarrow$ end path. It is important to note that we assume ridership is evenly distributed across a time period. We must make this assumption as our time period distribution (see Table 2) is the lowest time granularity that we have data for.

Moreover, to simplify the model, we do not distinguish the directionality of each line. Because the distribution of trains along the line will be the same for both directions (i.e. one way cant always have two times the number of trains as the other way because each train has to go back on the line), it simplifies the model if we find the total number of trains that need to travel the line and then assume that they are 50/50 split between each direction of the train line.

We seek to optimize the schedule over five different time periods. From `bart.gov`, we obtained the ridership distribution for March 2023 over the five different time periods displayed in Table 2. We then run five optimization problems, each one multiplying the ridership adjacency matrix by the percentage of people in the day who ride at that time period (rounding up all values). Because each time period elapses a different number of hours, we must account for this in our problem as well.

Time Period	Duration in Hours (h_t)	Total Ridership	Percent Total Ridership
Early AM	1	1774	1.17%
AM Peak	3	33940	22.45%
Mid-Day	7	50204	33.21%
PM Peak	3.5	48812	32.29%
Evening	1.5	16420	10.86%

Table 2: BART Ridership Data for each Time Period

4 Formalized Problem

Our formalized problem is as follows:

For t in time periods:

$$\text{Minimize} \quad \sum_{k=1}^{n} d_k \cdot \ell_k$$

$$\text{s.t.} \quad \sum_{(i,j):\ell_k \in B_{(i,j)}} A_{(i,j)} \leq 70 \cdot h_t \cdot t_k \cdot \ell_k \qquad (1 \leq k \leq n)$$

$$0.25 \cdot \ell_k \geq 2 \cdot t_k \qquad (1 \leq k \leq n)$$

$$\ell_k \geq 0 \qquad (1 \leq k \leq n)$$

$$\ell_k \in \mathbb{Z} \qquad (1 \leq k \leq n)$$

As explained in the previous section, we are running five different optimization for each t in time periods. We then minimize the distance of line k (d_k) multiplied by the number of train cars that are on line k (ℓ_k). It is important to note that a train can consist of multiple train cars, and ℓ_k represents the number of train cars. Our first constraint states that the total number of riders using a line must be less than or equal to the train capacity (70) times the number of runs a train can make within the time period ($h_t \cdot t_k$) times the number of trains on that line (ℓ_k). This is the ridership constraint. The left side of the first constraint is the sum of all entries in matrix A such that ℓ_k is contained in the list of the corresponding entry in matrix B.

The second constraint states that a train should arrive at a station every quarter hour. To model this, we originally formulated our constraint as having the total time it takes for a train to complete a full circuit (there and back) divided by the number of train cars be less than or equal to a quarter of an hour. Mathematically, we write the original constraint as follows:

$$\frac{2 \cdot t_k}{\ell_k} \leq 0.25$$

However, because Gurobi requires variables to be in the numerator, we used some simple math to rewrite the constraint to the form displayed in the formalized problem.

Additionally, we have a non-negative constraint for each line variable ℓ_k because there cant be a negative number of trains on a line. On the same note, each line variable ℓ_k must be an integer as it is not possible to have a fraction of a train car.

5 Code

Figure 2 shows the code we wrote to solve our model using the Gurobi Python API. The list l is the list containing all variables ℓ_k. You can see under the second comment that we set our objective function to the sum of the distance times the number of train cars for each line. We then added our constraints by transforming matrix B into a boolean numpy array which displayed true for an entry (i, j) if the current train line k was the list $B_{(i,j)}$. We then used this to find the total ridership of a line by summing A where B = **True**. Lastly we called the optimize function to run the problem and find the optimal results. As we only had 8 variables in our model, the problem ran very quickly - in under a second.

```
#Gurobi Model

model = Model('BART Optimization')
l = []

#For each line, create the variable l_x with lower bound 0
for i in milesPerLine[:,0]:
    l.append(model.addVar(vtype=GRB.INTEGER, lb=0, name='l{}'.format(i)))

#Objective function is sum of milesPerLine[i] * l[i]
model.setObjective(quicksum(milesPerLine[i,1]*l[i] for i in range(len(l))), GRB.MINIMIZE)


#Constraints
for k in range(1,len(l)+1):
    lineSum = 0
    for i in range(len(linedata)):
        ridedataPartition = ridedata_n[i]
        lineSum += np.sum(ridedataPartition[np.array([k in sublist for sublist in linedata[i]])])
    model.addConstr(70 * (h_t / lineRuntime[k-1, 1]) * l[k-1] >= lineSum)
    model.addConstr(0.25 * l[k-1] >= 2 * lineRuntime[k-1, 1])


model.optimize()
```

Figure 2: Code for the Gurobi Model

6 Results

Table 3 displays the results from each of the 5 optimization problems. As expected given transit systems are most often used as a means to commute to work, AM Peak and PM Peak are the time periods which need the highest number of trains. As noted earlier, these variables display the optimal number of train cars per line. For ℓ_2 at PM Peak, it takes 1.54 hours or 92.4 minutes to complete a full cycle (.77hrs to go one way, 1.54hrs to go there and back). This means with 118 individual trains a train would arrive at a station about every 47 seconds. What might happen in reality is that BART chooses each train to have 3 train cars for line 2, meaning that a train would arrive every 2.3 minutes instead.

Early AM		AM Peak		Mid-Day		PM Peak		Evening	
ℓ_1	21	ℓ_1	50	ℓ_1	31	ℓ_1	60	ℓ_1	51
ℓ_2	24	ℓ_2	97	ℓ_2	61	ℓ_2	118	ℓ_2	96
ℓ_3	6	ℓ_3	20	ℓ_3	13	ℓ_3	24	ℓ_3	20
ℓ_4	14	ℓ_4	30	ℓ_4	18	ℓ_4	36	ℓ_4	31
ℓ_5	3	ℓ_5	3	ℓ_5	3	ℓ_5	3	ℓ_5	3
ℓ_6	6	ℓ_6	8	ℓ_6	6	ℓ_6	9	ℓ_6	8
ℓ_7	1	ℓ_7	1	ℓ_7	1	ℓ_7	1	ℓ_7	1

Table 3: Results of the MILP for each time period

Its also interesting to note that line 7, the two-station line to the Oakland International Airport, can meet its ridership constraints and arrive at both of its stations every 15 minutes with only one train for every time period.

7 Next Steps

In order to improve the model's accuracy given more real-world constraints, we would want to look at the network as a station-to-station network modelling problem rather than a line-by-line problem. We would also want to look at line color rather than physical lines to make the problem more realistic as transit systems assign trains to color lines rather than physical lines. This might require more data on what stations each passenger goes through on their journey to pinpoint which stations people are likely to use to transfer lines. Lastly, it would be nice to get the passenger data broken out in more granular times, as that would enable us to create an exact train schedule for a weekday.