# Movement planner Algorithm for optimal railway scheduling on Multi-Track Territories

**Team : Xyon**

Pushkar Godbole

4th year UG Aerospace Engineer

Department of Aerospace Engineering

Indian Institute of Technology – Bombay

## 1. Introduction

The solution proposed is a 3 stage-stage approach : The first stage involves intelligent greedy routing of the given trains. The second stage consists of MIP based optimal scheduling of the above routed trains. The third stage imposes an iterative path swapping and optimal scheduling heuristic to improve the solution. Additionally, to augment scalability the above 3-stage approach is nested inside a sub-problem generation block to deal with large problem sizes.

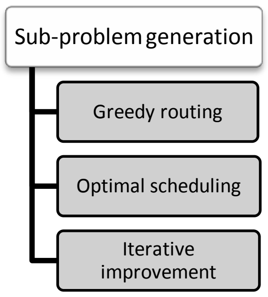


**Figure 1 : Overall solution approach**

## 2. Train Routing

All the territory and train data from the input file is read into suitable data-structures. Train routing is then achieved in the following manner.

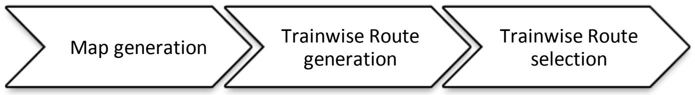


**Figure 2 : Train routing approach**

### 2.1. Map generation

The fact that all the arc data is given from left to right is made use of in the map generation. First the arcs are sorted w.r.t. their node-1 using merge-sort. The map is then generated and stored in a vector of vectors of arcs with one vector of arcs corresponding to each node. Each of such vectors contains the forward connected arcs from that node.

Toy example map :

```
| 0 : (0,1) | 1 : (1,3) (1,2) | 2 : (2,4) | 3 : (3,5) | 4 : (4,5) | 5 : (5,6) |
| 6 : (6,7) | 7 : (7,9) (7,8) | 8 : (8,11) | 9 : (9,10) | 10 : (10,11) | 11 : (11,12) | | 12 : ~ |
```

### 2.2. Trainwise Route Generation

Depth first search is employed for route generation. First, all the required origin - destination pairs are identified from the train data to avoid repetition. All paths between the origin and destination are generated from the map using depth-first search for each origin - destination pair. Every new path found during the depth first search is associated with a path-id. This path-id uniquely identifies a path w.r.t. the sidings and crossovers contained in it.

The path-id is generated as follows :

- Path id starts with `'id#'`
- If a path contains siding(s), an identifier of format `'S<node1>-<node2>:'` gets added for every siding in the path.
- After all sidings have been added, if a path contains crossover(s), an identifier of format `'<node1>-<node2>:'` gets added for every crossover in the path.

The sidings and crossovers are added to the path-id from left to right w.r.t. the map irrespective of the path direction or origin - destination pair. So a path from node 0 to node 39 which takes siding : 16-18 and crossover : 37-41 will have a path-id `'id#S16-18:37-41:'`. And a path from node 80 to node 0 which takes crossover : 41-37 and siding : 18-16, (that is essentially the same path) will also have the same path-id.

It would be very inefficient for a train to take more than 1 siding in its entire travel. Hence to eliminate this inefficiency and to reduce the size of the path pool, all paths with multiple sidings are eliminated.

Then, all the filtered paths with appropriate origin-destination pair are inserted into the path pools of the corresponding trains. After all paths have been inserted trainwise, following check is performed for each train :

- If the train carries ‘Hazardous materials’, all paths with sidings are eliminated
- If the train length exceeds a siding length, all paths containing that siding are eliminated
- All paths missing a scheduled arrival node of the train are eliminated

### 2.3. Trainwise Route selection

The new filtered path pools for each train are then doubly sorted first w.r.t. to ‘Un-preferred track usage’ and then internally w.r.t. the total number of sidings+crossovers in the path, using merge-sort. For example, for a train with the following paths, the sorted path pool will be :

Path 1 – Un-preferred track usage = No Sidings = 0 Crossovers = 0
Path 2 – Un-preferred track usage = No Sidings = 1 Crossovers = 0
Path 3 – Un-preferred track usage = Yes Sidings = 0 Crossovers = 1
Path 4 – Un-preferred track usage = Yes Sidings = 1 Crossovers = 1

Once the individual path pools for all trains are sorted, the trains are then triply sorted first w.r.t. ‘Tons per operative brake’ exceeding 100 and then internally w.r.t. the train type (A/B/C/D/E/F in that order) and finally w.r.t. train entry time.
For example, for the following 6 trains, the sorting will be :

B3 – TOB = 125 ( > 100) Train type = B Entry time = 500sec
A2 – TOB = 75 ( < 100) Train type = A Entry time = 0sec
A1 – TOB = 75 ( < 100) Train type = A Entry time = 200sec
B1 – TOB = 75 ( < 100) Train type = B Entry time = 100sec
B2 – TOB = 75 ( < 100) Train type = B Entry time = 400sec
C1 – TOB = 80 ( < 100) Train type = C Entry time = 0sec

With the pathwise and trainwise sorting done, the trains are now assigned paths in the aforementioned sorted order.
At every step of the assignment, the assigned path is marked as taken and cannot be taken by any other train later except TOB > 100 trains. This is facilitated by recording the unique path-id corresponding to that path after every assignment.

- The TOB > 100 trains are invariably assigned their first path i.e. the path without any siding/crossover.
- The subsequent trains are assigned the first un-assigned path in their path pools.

Thus, the priority (SA and TOB > 100) trains tend to be assigned better paths over the low priority trains. Amongst the same type trains, trains with earlier entry time tend to be assigned better paths.

## 3. The MIP model and optimal scheduling

The optimal scheduling problem of the routed trains is modeled as an MIP. The objective of the MIP is to minimize the cost function :

Total cost = (Total Delay × Delay Penalty/Hour) +
(Schedule deviance over 2 hours for SA Trains × Penalty over 2-hour deviance/Hour) +
(Terminal Want Time deviance beyond the 4-hour window × Penalty for TWT/Hour) +
(Un-preferred Track Time × Penalty for Un-preferred Track Utilization/Hour)

The primary variables to be optimized are the entry times of every train at each node in its path subject to following constraints :

$t_{pq}$ : Entry time of train no. $p$ in the node no. $q$ of its path, node no. 1 being its origin
$N$ : Total number of trains in consideration for the MIP

## 3.1. Train entry time at origin

$t_{p1} \geq max(Train\ p\ entry\ time,\ Scheduling\ start\ horizon) \qquad p \in (1,N)$ (1)

*Scheduling start horizon* is 0 for the first sub-problem and end time of the preceding sub-problem for subsequent sub-problems.

## 3.2. Train entry time at subsequent nodes

This constraint sets the lower bound on time required for a train to traverse an arc based on the arc length and train speed on that arc. The upper bound for this constraint ideally is ∞ as a train is allowed to wait at any arc in its path. But setting upper bound to ∞ for every arc unnecessarily increases the problem size. Hence upper bound is set to ∞ only for arcs where a train might require to wait.

These include :

a) Before origin
b) Arc just after every schedule adherence node
c) Arc 1 prior to every convergent node w.r.t train direction for meet pass events

Waiting at all other arcs would be equivalent to waiting at one of the above arcs w.r.t the cost and hence can be ignored.

For example, for a train with origin 0 and destination 12, with scheduled arrival at node 6 and node 12 on the toy example track the waiting arcs should only be :

a) Before node 0
b) Arc 6-7
c) Arc 1 prior to convergent node 5 : arc 1-3, arc 2-4
   Arc 1 prior to convergent node 11 : arc 7-8, arc 9-10
   *For a train from node 12 to node 0, this will change to :*
   Arc 1 prior to convergent node 7 : arc 11-8, arc 10-9
   Arc 1 prior to convergent node 1 : arc 5-3, arc 4-2

$t_{p(q+1)} - t_{pq} \geq req\_t_{pq}$ For above 3 cases (2a)

*or*

$t_{p(q+1)} - t_{pq} = req\_t_{pq}$ For rest of the arcs (2b)

$req\_t_{pq}$ : Minimum time required for train $p$ to traverse arc $q$. Note that the number of arcs is 1 less than the number of $t_p s$.

Now, every equality constraint reduces the number of $t$ variables by 1.

## 3.3. MOW constraint

If arc $q$ of train $p$ is same as arc of $mow_i$ :

$-\ mow\_x_i \times M + t_{pq} \leq mow\_starttime_i$ (3a)

$(1 - mow\_x_i) \times M + t_{pq} \geq mow\_endtime_i$ (3b)

$mow\_x_i \in \{0,1\}$

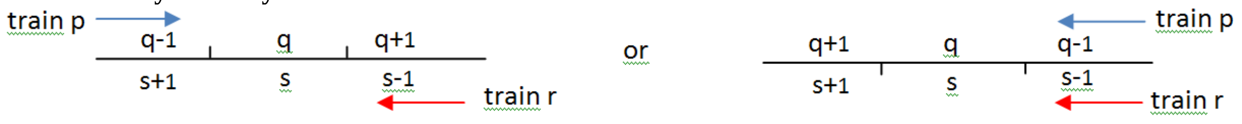
$M$ is a very large number here taken as 100000

## 3.4. Non-concurrency/Headway constraint

The non-concurrency constraint is just a subset of headway constraint. Implementing the headway constraint will automatically take care of the non-concurrency constraint.

$x$ : Headway precedence decision variable

### 3.4.1. Primary headway constraint



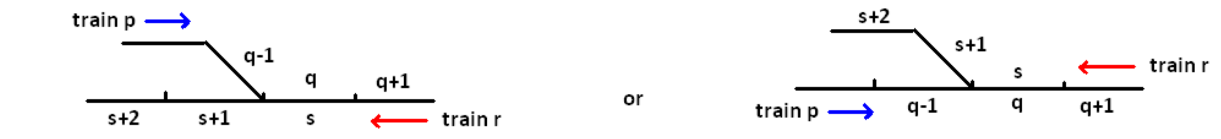
$x_j \times M + t_{pq} - t_{r(s+1)} \geq 300\ (5\ minutes)$ (4a)

$(1 - x_j) \times M + t_{rs} - t_{p(q+1)} \geq 300\ (5\ minutes)$ (4b)

$x_j \in \{0,1\}$

Besides these primary headway constraints, some additional headway constraints need to be imposed for special cases.

### 3.4.2. Diverging Meet-pass headway constraint

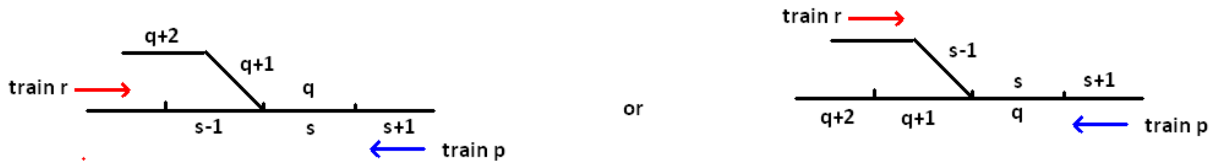


For above case :

$x_k \times M + t_{p(q-1)} - t_{r(s+2)} \geq 300$ *(5 minutes)* (5a)

$(1 - x_k) \times M + t_{rs} - t_{p(q+1)} \geq 300$ *(5 minutes)* (5b)

$x_k \in \{0,1\}$



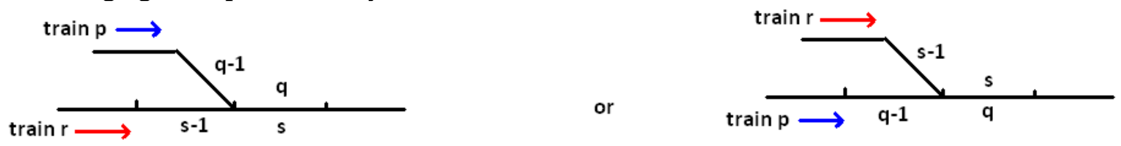
For above case :

$x_l \times M + t_{r(s-1)} - t_{p(q+2)} \geq 300$ *(5 minutes)* (6a)

$(1 - x_l) \times M + t_{pq} - t_{r(s+1)} \geq 300$ *(6b)* (6b)

$x_l \in \{0,1\}$

### 3.4.3. Converging Meet-pass headway constraint



For above case :

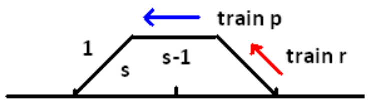
$x_m \times M + t_{p(q-1)} - t_{rs} \geq 300$ *(5 minutes)* (7a)

$(1 - x_m) \times M + t_{r(s-1)} - t_{pq} \geq 300$ *(5 minutes)* (7b)

$x_m \in \{0,1\}$

### 3.4.4. Mid-start headway constraint

This constraint is imposed for trains which have an origin in the middle of the map. The trains are thus already occupying an arc preceding their entry node. Let *p* be such a train and *r* be another train having an arc overlapping with *p's* entry arc. Thus *r* cannot enter its arc *(s-1)* until 5 minutes post *p* entering its arc 1.



$t_{r(s-1)} - t_{p1} \geq 300$ *(5 minutes)* (8)

## 3.5. Penalty constraints

These constraints are imposed to correctly calculate the cost function while maintaining the linearity of the model

$hor\_x_{pq}$ : Horizon crossing detection variable at arc *q* for train *p*

### 3.5.1. Horizon crossing detection

$-(1 - hor\_x_{pq}) \times M + t_{pq} \leq 43200$ *(12 hours)* (9a)

$hor\_x_{pq} \times M + t_{pq} \geq 43200$ *(12 hours)* (9b)

$hor\_x_{pq} \in \{0,1\}$

$hor\_x_{pq} = 1$ when $t_{pq}$ is within the planning horizon and 0 otherwise

### 3.5.2. Delay penalty

$d_{pq}$ : Delay at arc $q$ for train $p$ within planning horizon

$d\_horslack_{pq}$ : Slack variable for delay at arc $q$ for train $p$ beyond planning horizon

$d_{pq} \geq 0$ (10a)

$d\_horslack_{pq} \geq 0$ (10b)

$d_{pq} + d\_horslack_{pq} = t_{p(q+1)} - t_{pq} - req\_t_{pq}$ (10c)

$- hor\_x_{pq} \times M + d_{pq} \leq 0$ (10d)

$- (1- hor\_x_{pq}) \times M + d\_horslack_{pq} \leq 0$ (10e)

$d_{pq}$ becomes 0 when $hor\_x_{pq}$ is 0 irrespective of the delay

### 3.5.3. Schedule adherence penalty

$sa$ : Schedule adherence delay beyond 2 hours but within planning horizon

$sa\_slack$ : Slack variable for schedule adherence delay within 2 hours and within planning horizon

$sa\_horslack$ : Slack variable for schedule adherence delay beyond planning horizon

$sa\_time$ : Prescribed schedule adherence time (given)

$sa\_x$ : Beyond 2 hours delay detection variable

$sa_{pq} \geq 0$ (11a)

$sa\_slack_{pq} \leq 0$ (11b)

$sa\_horslack_{pq} \geq 0$ (11c)

$sa_{pq} + sa\_slack_{pq} + sa\_horslack_{pt} = t_{pq} - sa\_time_{pq} - 7200 \text{ (2 hours)}$ (11d)

$sa\_x_{pq} \times M + t_{pq} - sa\_time_{pq} \geq 7200 \text{ (2 hours)}$ (11e)

$- (1- sa\_x_{pq}) \times M + t_{pq} - sa\_time_{pq} \leq 7200 \text{ (2 hours)}$ (11f)

$sa\_x_{pq}$ is 0 when schedule adherence delay is beyond 2 hours and 1 otherwise

$sa\_x_{pq} \times M + sa\_slack_{pq} \geq 0$ (11g)

$- (1- sa\_x_{pq}) \times M + sa_{pq} \leq 0$ (11h)

$sa\_x_{pq} \in \{0,1\}$

$- hor\_x_{pq} \times M + sa_{pq} \leq 0$ (11i)

$- (1- hor\_x_{pq}) \times M + sa\_horslack_{pq} \leq 0$ (11j)

$sa_{pq}$ becomes 0 when $hor\_x_{pq}$ is 0 irrespective of the delay

### 3.5.4. Terminal want time penalty

Terminal want time penalty has been split into two parts. Upper TWT penalty is applied if the train reaches the terminal beyond 3 hours of the TWT, where as lower TWT penalty is applied if the train reaches the terminal before 1 hour of the TWT.

$twtu$ : Terminal want time delay beyond 3 hours but within planning horizon

$twtu\_slack$ : Slack variable for terminal want time delay within 3 hours and within planning horizon

$twtu\_horslack$ : Slack variable for terminal want time delay beyond planning horizon

$twtl$ : Terminal want time furtherance over 1 hour and within planning horizon

$twtl\_slack$ : Slack variable for terminal want time furtherance within 1 hour and within planning horizon

$twtl\_horslack$ : Slack variable for terminal want time furtherance beyond planning horizon

$twt\_time$ : Prescribed terminal want time (given)

$twt\_xu$ : Beyond 3 hours delay detection variable

$twt\_xl$ : Over 1 hour furtherance detection variable

$twtu_{pq} \geq 0$ (12a)

$twtu\_slack_{pq} \leq 0$ (12b)

$twtu\_horslack_{pq} \geq 0$ (12c)

$twtu_{pq} + twtu\_slack_{pq} + twtu\_horslack_{pq} = t_{pq} - twt\_time_{pq} - 10800 \text{ (3 hours)}$ (12d)

$- twt\_xu_{pq} \times M + t_{pq} - twt\_time_{pq} \leq 10800 \text{ (3 hours)}$ (12e)

$(1- twt\_xu_{pq}) \times M + t_{pq} - twt\_time_{pq} \geq 10800 \text{ (3 hours)}$ (12f)

$twt\_xu_{pq}$ is 1 when schedule adherence delay is beyond 3 hours and 0 otherwise

$(1 - twt\_xu_{pq}) \times M + twtu\_slack_{pq} \geq 0$ (12g)

$- twt\_xu_{pq} \times M + twtu_{pq} \leq 0$ (12h)

$twt\_xu_{pq} \in \{0,1\}$

$- hor\_x_{pq} \times M + twtu_{pq} \leq 0$ (12i)

$- (1- hor\_x_{pq}) \times M + twtu\_horslack_{pq} \leq 0$ (12j)

$twtu_{pq}$ becomes 0 when $hor\_x_{pq}$ is 0 irrespective of the delay

$twtl_{pq} \geq 0$ (13a)

$twtl\_slack_{pq} \leq 0$ (13b)

$twtl\_horslack_{pq} \geq 0$ (13c)

$twtl_{pq} + twtl\_slack_{pq} + twtl\_horslack_{pq} = twt\_time_{pq} - t_{pq} - 3600\ (1\ hour)$ (13d)

$- twt\_xl_{pq} \times M + twt\_time_{pq} - t_{pq} \leq 3600\ (1\ hour)$ (13e)

$(1- twt\_xl_{pq}) \times M + twt\_time_{pq} - t_{pq} \geq 3600\ (1\ hour)$ (13f)

$twt\_xl_{pq}$ is 1 when schedule adherence furtherance is over 1 hour and 0 otherwise

$(1 - twt\_xl_{pq}) \times M + twtl\_slack_{pq} \geq 0$ (13g)

$- twt\_xl_{pq} \times M + twtl_{pq} \leq 0$ (13h)

$twt\_xl_{pq} \in \{0,1\}$

$- hor\_x_{pq} \times M + twtl_{pq} \leq 0$ (13i)

$- (1- hor\_x_{pq}) \times M + twtl\_horslack_{pq} \leq 0$ (13j)

$twtl_{pq}$ becomes 0 when $hor\_x_{pq}$ is 0 irrespective of the delay

#### 3.5.5. Un-preferred track usage

Un-preferred track usage penalty is calculated only for arcs in the un-preferred region of the path. These also include the sidings switch tracks connected to the un-preferred main track.

*utu* : Un-preferred track usage within planning horizon

*utu_horslack* : Slack variable for un-preferred track usage beyond planning horizon

$utu_{pq} \geq 0$ (14a)

$utu\_horslack_{pq} \geq 0$ (14b)

$utu_{pq} + utu\_horslack_{pq} = t_{p(q+1)} - t_{pq}$ (14c)

$- hor\_x_{pq} \times M + utu_{pq} \leq 0$ (14d)

$- (1- hor\_x_{pq}) \times M + utu\_horslack_{pq} \leq 0$ (14e)

$utu_{pq}$ becomes 0 when $hor\_x_{pq}$ is 0 irrespective of the delay

After implementing all the constraints, the cost function is calculated from *d, sa, twtu, twtl,utu* by multiplying them with their appropriate penalty multipliers.

### 3.6. Redundant siding elimination

After the optimal cost is calculated from the MIP, the solution is very likely to have paths with redundant sidings on account of the distinct paths allotted to every train. A siding can be said to be in usage if :

- The train itself waits at the siding
- The train passes through the siding as another train is waiting on the corresponding main track arc at that time

In all other cases, the siding usage can be considered redundant and eliminated. The train paths are checked for such cases and a path excluding the siding is allotted those trains using redundant sidings. This is again facilitated by the unique path-id characterized by siding/crossover usage allotted to the train.

The MIP optimizer is again run with the new train-path configuration. The solution obtained by this process although not globally optimal was always found to be very close to global optimality.

This precise solution is then output to the terminal and the redundant and the precise solutions are fed to the iterator for improvement.

## 4. Iterative improvement

The redundant solution found in the previous block is iteratively improved in this block. This is done by randomly swapping two paths amongst trains and checking for improvement.

The process followed is :

- Randomly choose 2 trains
- Check if their paths can be swapped by looking for train 2's path id in the path pool of train 1 and vice-versa.
- Once found check if the newly generated combination of train paths has already been explored
- Continue this process until a new combination of train paths is found or the upper-bound on the number of search iterations is reached

Once found, this new train path combination is again optimized by the MIP optimizer. If the cost given by this combination is less than the previous best redundant solution, the MIP optimizer is again run on the concise (without redundant sidings) counterpart of the combination. If this solution is better than the previous best concise solution, it is accepted and the output to the terminal.
This process is continued until an upper bound on the number of iterations or possible combinations is reached.

## 5. Dealing with large problems

It was observed that solving the MIP for more than 15 trains at a time was very time intensive. Good quality solutions were found for problems upto 12 trains within acceptable amount of time. To deal with this issue, the algorithm splits large problems into solvable sub-problems.
This is done as follows :

- If the problem size is less than or equal to 12 trains, set chunk size to 12 and solve it as a single problem
- If the problem size is beyond 12 trains, check the number of SA trains. If this number is between 12 and 15, set the chunk size to the number of SA trains
- If number of SA trains is less than 12, set chunk size to 12 and if number of SA trains is more than 15, set chunk size to 15.

Chunk size is the maximum number of trains to be considered in a sub-problem.
With this method, the algorithm attempts to split large problems into two chunks of priority (SA + TOB > 100) trains and low priority (NSA) trains. In case of very large problem sizes, the chunk size is invariably set to 15. Thus the priority trains are scheduled first and the cost efficient (low priority) trains are pushed to the back. Since the trains have already been sorted w.r.t. priority, they are serially picked into the sub-problems.
Every sub-problem takes the end-time data of the last train from the previous sub-problem as its starting horizon. The starting horizon for the first sub-problem is 0.
The entire routing and scheduling block is nested inside this sub-problem generation block.

The output file is generated at the end of the code run.

## 6. Execution and Computational results

All codes were written in C++ and the MIP optimizer used was ILOG CPLEX 12. The codes were run on IIT-Bombay's IEOR server 'Optimus' with : Fedora 14, Intel X4300 M3, Quad core Xeon E5506 and 64GB RAM.
An upper bound of 150 seconds was set on all MIP optimization instances to protect the execution from stagnating. Multiple redundancies have been incorporated in the code to protect it from errors. The iterative improvement block is run 8 times. This was set by considering the Data set 3 input file as a benchmark problem w.r.t. runtime and solution quality.
The program can be run by compiling and executing the file 'ras_main.cpp' on a machine with ILOG CPLEX 12 or higher. A compilation script 'runCplex.sh' is also included which can be executed as follows :

```
./runCplex.sh c++ ras_main
```

The script may require editing to change the address of the cplex libraries. Compilation will generate an executable shell file named 'ras_main'.
On execution, the code asks for an input file. The input file name should be devoid of spaces.

For example, to execute the program for DATASET 1, the file name and hence the input argument must be : 'RAS_DATASET1.txt'. An output text file in xml format with 'OUTPUT' appended at end of its name would be generated. The input file needs to be placed in the same directory as the ras_main file.

The computational results have been summarized in the following table :

| **Problem set** | **Chunk size** | **Number of sub-problems** | **Final Cost** | **First solution (solution to sub-problem 1)** | **Cost of first solution (solution to sub-problem 1)** | **Run time** |
|---|---|---|---|---|---|---|
| Toy example | 12 | 1 | 762.68 $ | 0.13sec | 762.68 $ | 0.13 sec |
| Data set 1 | 12 | 1 | 1967.29 $ | 290 sec | 3215.94 $ | 11 mins |
| Data set 2 | 13 | 2 | 4108.95 + 8400 = 12508.95 $ | 265 sec | 4108.95 $ | 21 mins |
| Data set 3 | 14 | 2 | 3770.51 + 9600 = 13370.51 $ | 250 sec | 3770.51 $ | 20 mins |

## 7. Concluding remarks

While designing the algorithm, closer attention has been paid to maintaining a balance between scalability and solution quality. It was observed that maintaining the linearity of the MIP substantially speeds up the process. The iterative improvement block introduces some amount of explorability in the algorithm over the initial greedy routing. The sub-problem generation is implemented to take care of the cost intensive trains first holding the low priority trains for later.