MIE562 Project - Truck Scheduling - MIP Model

Team 5 - Dylan Camus, Ryan Do, Fan Jia, Matheus Magalhaes, Sugumar Prabhakaran

Date: 8 Nov 2020

In [1]:
```
import numpy as np
import pandas as pd
import plotly.express as px

import gurobipy as gp
from gurobipy import Model, GRB, quicksum, max_
```

Introduction

1. Sets

- $k \in K$: A container, k in a set of containers, K (Ex. $K = \{1, 2, 3, \ldots\}$).
- $j \in J$: A carrier, j, in a set of carriers, J (Ex. $J = \{1, 2, 3, \ldots\}$).
- $c \in C$: A chassis, c, in a set of chassis, C (Ex. $C = \{1, 2, 3, \ldots\}$).
- $L = \{1, 2, 3\}$: is the set of travel legs, where:
  - 1 : terminal to transloading facility leg
  - 2 : terminal to stack leg
  - 3 : stack to transloading facility leg

2. Parameters

- $n_k = |K|$: number of containers
- $n_j = |J|$: number of carriers
- $n_c = |C|$: number of chassis
- S: fixed cost at stack [$/container]
- S': variable cost at stack [($/day)/container]
- R_k: Release date for container k [days]
- P_k: Transload processing time for container k [days]
- ρ_k: priority factor of container k (higher value is higher priority)
- Y_{jk}: binary parameter = 1 if container k belongs to carrier j
- T'_j: Demurrage free period before cost for carrier j at terminal [days]
- T_j: Demurrage daily cost per container for carrier j at terminal [($/days)/container]
- δ'_j: free period before detention cost for containers of carrier j [days]
- δ_j: detention cost/unit time for containers of carrier j [($/days)/container]

- Φ_{kl}: Time duration transporting container k on leg l, which includes a variable processing time for legs 1 and 3 [days]
- $D_{ll'}$: Delay required for a chassis to start leg l' after completing l (including travel) [days]
- M: Some large number

In [2]:

```
def read_instance(instance_csv_path: str):
    '''
    reads instance file into a pandas dataframe for easy viewing

    ARGS:
    (str) instance_csv_path:  file name stored in variable

    RETURN:
    (df) df:     output a pandas dataframe with appropriate rows
    '''
    df = pd.read_csv(instance_csv_path) # read csv file into dataframe

    #rename dataframe rows:
    df.rename(index={0:"header_data",
                     1:"container (k)",
                     2:"carrier (j)",
                     3:"rel_date (R_k)",
                     4:"proc_time (P_k)",
                     5:"priority (rho_k)",
                     6:"free_demurrage (T'_j)",
                     7:"demurrage_cost (T_j)",
                     8:"free_detention (delta'_j)",
                     9:"detention_cost (delta_j)",
                     10:"leg_travel_times"}, inplace=True)
    return df
```

In [3]:

```
def generate_parameters(df):
    '''
    generate all necessary parameters from instance file that is read into a df

    ARGS:
    (df) df:        a pandas dataframe in specific format containing info

    RETURN:
    (int) nK:            number of containers
    (int) nJ:            number of carriers
    (int) nC:            number of chassis
    (int) S:             fixed cost at stack ($)
    (int) S_prime:       variable daily cost per container at stack ($)
    (list) K:            list of containers
    (list) J:            list of carriers
    (list) C:            list of chassis
    (list) L:            list of legs
    (dict) R:            release date for container k (key)
    (dict) p:            processing time at transload for container k (key)
    (dict) rho:          priority factor for container k (key)
    (dict) Y:            binary value of 1 if container k belong to carrier j (key)
    (dict) T_prime:      demurrage free period per carrier j (key)
    (dict) T:            demurrage cost/day/container per carrier j (key)
    (dict) Delta_prime:  detention free period per carrier j (key)
    (dict) Delta:        detention cost/day/container per carrier j (key)
    (dict) Phi:          Time duration container k on leg l, incl. proc. time for legs 1 ar
    (dict) D:            Delay required for chassis to start next leg
    (int) M:             Some large constant
    '''

    #header data from instance: (nk, nJ, nC, S, S_prime):
    nK, nJ, nC, S, S_prime = tuple((df.iloc[0].dropna()).astype(float).astype(int))
```

```
    #create sets K, J, C and L:
    K = [k for k in df.iloc[1].dropna().astype(float).astype(int)]
    assert nK >= len(K)
    J = list(set(df.iloc[2].dropna()))
    assert nJ >= len(J)
    C = list(range(nC))
    L = [1, 2, 3]

    #Y stores carrier j for each container k:
    Y = {j:{k: 0 for k in K} for j in J}
    for i, k in enumerate(K):
        Y[df.iloc[2].dropna()[i]][k] = 1

    #read rows 3 to 5 from df to R, p, rho for each container k:
    R = {k: int(df.iloc[3].dropna()[i]) for i, k in enumerate(K)}
    p = {k: int(float(df.iloc[4].dropna()[i])) for i, k in enumerate(K)}
    rho = {k: int(df.iloc[5].dropna()[i]) for i, k in enumerate(K)}

    T_prime = {df.iloc[6].dropna()[i]: int(df.iloc[6].dropna()[i+1])
               for i in range(0, len(df.iloc[6].dropna()), 2)}
    T = {df.iloc[7].dropna()[i]: int(df.iloc[7].dropna()[i+1])
         for i in range(0, len(df.iloc[7].dropna()), 2)}
    Delta_prime = {df.iloc[8].dropna()[i]: int(df.iloc[8].dropna()[i+1])
                   for i in range(0, len(df.iloc[8].dropna()), 2)}
    Delta = {df.iloc[9].dropna()[i]: int(df.iloc[9].dropna()[i+1])
             for i in range(0, len(df.iloc[9].dropna()), 2)}

    #generate Phi from leg_times and processing times for each container k:
    leg_times = {l: float(df.iloc[10][i*2+1]) for i, l in enumerate(L)}
    Phi = {k: {1: 2*leg_times[1] + p[k], 2: leg_times[2],
               3: leg_times[3] + p[k] + leg_times[1]} for k in K} # duration of leg l (inc

    D = {1:{1: 0, 2: 0, 3: leg_times[2]},
         2:{1: leg_times[2], 2: leg_times[2], 3: 0},
         3:{1: 0, 2: 0, 3: leg_times[2]}}
    M = 1000   #some large integer

    return nK, nJ, nC, S, S_prime, K, J, C, L, R, p, rho, Y, T_prime, T, Delta, Delta_prim
```

3. Decision Variables

- Binary Variable: $x_{klc} = \begin{cases} 1, & \text{if container k travels leg l on chassis c} \\ 0, & \text{otherwise} \end{cases}$
- Start time s of container k on leg l on chassis c: s_{klc}, where $s_{klc} \geq 0$
- Binary Variable: $z_{klk'l'c}$
  $= \begin{cases} 1, & \text{if transporting container } k' \text{ on chassis c along leg } l' \text{ is scheduled after transporting conta} \\ 0, & \text{otherwise} \end{cases}$

4. Objective Function

We want to minimize the Total Cost (c) = (Demurrage cost) + (Detention cost) + (Stack cost) + (Priority penalty cost)

- **objective function:** $min\ (c)$
- $c = \text{demurrage cost} + \text{detention cost} + \text{stack cost} + \text{priority cost}$

(1) Demurrage Cost:

The Demurrage Cost refers to cost associated with keeping containers in the terminal beyond the free period allotedin days (T_j'') that is different for each carrier.

If the start time for a container (s_{klc}) is higher than the release date (R_k) + the free period (T_j''), there will be a variable cost per extra day (T_j) per container.

- $\text{demurrage cost} = \sum_{k\in K}\sum_{c\in C}\sum_{l\in\{1,2\}}\sum_{j\in J} T_j x_{klc} Y_{jk} \cdot max\left[(s_{klc} - R_k - T_j''), 0\right]$

(2) Detention Cost

The terminal also charges a detention cost for containers that are not returned to the terminal for processing before return to the carrier. Similar to demurrage, there is a free period before which the containers must be returned to the terminal (δ_j'').

If the start time for a container (s_{klc}) and the total travel time of a container back to the terminal including processing time (ϕ_{kl}) is greater than the release date (R_k) + the free period (δ_j''), there will be a variable cost per extra day (T_j) per container.

- $\text{detention cost} = \sum_{k\in K}\sum_{c\in C}\sum_{l\in\{1,3\}}\sum_{j\in J} \delta_j x_{klc} Y_{jk} \cdot max\left[(s_{klc} + \phi_{kl} - (R_k + \delta_j'')), 0\right]$

(3) Stack Storage Cost

The stack cost (c_{stk}) is the sum of the fixed cost percontainer (S) and the variable cost per day per container (S') x the number of days that a container is in the stack, which is the difference between leg 3 and leg 2 start times (s_{klc}).

- $\text{stack cost} = \sum_{k\in K}\sum_{c\in C} x_{k2c} \cdot (S + S'(s_{k3c} - (s_{k2c} + \phi_{k2})))$

(4) Priority Penalty Cost

The priority penalty cost (c_{pri}) is applied for every day that a container sits in the terminal beyond the release day (R_k). Since the priority factor per container (ρ_k) is higher for higher priority containers, the associated penalty costs will be higher.

- $\text{priority cost} = \sum_{k\in K}\sum_{c\in C}\sum_{l\in\{1,2\}} \rho_k \cdot (s_{klc} - R_k) \cdot x_{klc}$

5. Constraints

(1) Resource constraints

For each chassis, transport jobs can't overlap and chassis (resources) may have to travel between jobs. For the case where job klc scheduled after job $k'l'c$, i.e. $s_{klc} > s_{k'l'c}$:

- $$M(1 - x_{klc}) + M(1 - x_{k'l'c}) + (s_{klc} - s_{k'l'c}) \geq \phi_{k'l'} + D_{l'l} - M \cdot z_{klk'l'c}, \quad \forall k, k' \in K, l, l' \in L, c \in C$$
$$s.t.\ (k = k' \neg \wedge l = l') \wedge k \leq k'$$

For the case where job $k'l'c$ scheduled after job klc, i.e. $s_{klc} > s_{k'l'c}$:

- $$M(1 - x_{klc}) + M(1 - x_{k'l'c}) + (s_{k'l'c} - s_{klc}) \geq \phi_{kl} + D_{ll'} - M \cdot (1 - z_{klk'l'c}), \quad \forall k, k' \in K, l, l' \in L,$$
$$s.t.\ (k = k' \neg \wedge l = l') \wedge k \leq k'$$

(2) Release date constraint

Start time (s_{klc}) must be after release date (R_k) for every container that leaves terminal on either leg 1 or leg 2 (ie. $x_{klc} = 1$):

- $s_{klc} \geq R_k x_{klc}, \quad \forall k \in K, l \in L, c \in C$

(3) Containers going through leg 2 must go through leg 3

Each container k will have a x_klc value of 1 or 0 for both leg 2 and leg 3 depending on if they travel through them or not:

- $\sum_{c \in C}(x_{k2c}) = \sum_{c \in C}(x_{k3c}), \quad \forall k \in K$

(4) Container must go through leg 1 OR (leg 2 AND 3)

Each container (k) will have $x_k lc = 1$ for either leg 1 or leg 2 so the sum of the two must be 1 for all containers (k) on all chassis (c).

- $\sum_{c \in C}(x_{k1c}) + \sum_{c \in C}(x_{k2c}) = 1, \quad \forall k \in K$

(5) Precedence Constraint: Leg 2 must be before Leg 3

Each container k will have a x_{klc} value of 1 for either leg 1 or leg 2 so the sum of the two must be 1 for all containers (k) on all chassis (c).

- $M \sum_{c \in C}(x_{k1c}) + \sum_{c \in C}(s_{k3c}) \geq= \sum_{c \in C}(s_{k2c}) + \Phi_{k2}, \quad \forall k \in K$

(6) Bind x=0 to s=0

When $x_{klc} = 0$, $s_{klc} = 0$

- $M x_{klc} \geq s_{klc}, \quad \forall k \in K, l \in L, c \in C$

(7) Domain of s_{klc}

The start time (s_{klc}) must be greater than 0:

- $s_{klc} \geq 0, \quad \forall k \in K, l \in L, c \in C$

In [4]:
```
def run_gurobi_model(S, S_prime, K, J, C, L, R, p, rho, Y, T_prime, T, Delta, Delta_prime,
                     LogToConsole = False, TimeLimit=60):
    '''
    run gurobi model on parameters generated from a test instance and produce best schedul
    obtainable in 60 seconds

    ARGS:
    all parameters generated from reading instance csv file (see generate_parameters funct

    RETURN:
    (dict) x:                binary 1 if container k is transported on leg l on chassis c
    (dict) s:                start time of container k on leg l on chassis c if it occurs
    '''

    #Create a new model
    model = gp.Model("Truck_Scheduling")
    model.setParam('TimeLimit', TimeLimit); # seconds
    model.params.LogToConsole = LogToConsole
```

```python
#Create decision variables
x = model.addVars(K, L, C, vtype=GRB.BINARY, name="x_klc")
s = model.addVars(K, L, C, vtype=GRB.CONTINUOUS, name="s_klc")
z = model.addVars(K, L, K, L, C, vtype=GRB.BINARY, name="z_klk'l'c")

#Max variables
max_expression_resource_constraint_1 = model.addVars(
    K, L, K, L, C, vtype=GRB.CONTINUOUS, name="max resource a")
max_expression_resource_constraint_2 = model.addVars(
    K, L, K, L, C, vtype=GRB.CONTINUOUS, name="max resource b")
max_expression_demurrage = model.addVars(
    K, L, C, J, vtype=GRB.CONTINUOUS, name="max_expression_demurrage")
max_expression_detention = model.addVars(
    K, L, C, J, vtype=GRB.CONTINUOUS, name="max_expression_detention")

# demurrage cost days max calculated for objective function:
model.addConstrs(max_expression_demurrage[k,l,c,j] >=
                 (s[k,l,c] - R[k] - T_prime[j])
                 for k in K for c in C for l in [1, 2] for j in J)
model.addConstrs(max_expression_demurrage[k,l,c,j] >= 0
                 for k in K for c in C for l in [1, 2] for j in J)

# detention cost days max calculated for objective function:
model.addConstrs(max_expression_detention[k,l,c,j] >=
                 (s[k,l,c] + Phi[k][l] - (R[k] + Delta_prime[j]))
                 for k in K for c in C for l in [1, 3] for j in J)
model.addConstrs(max_expression_detention[k,l,c,j] >= 0
                 for k in K for c in C for l in [1, 3] for j in J)

#Define objective function elements:
demurrage_cost = quicksum(x[k,l,c]*T[j]*Y[j][k]*max_expression_demurrage[k,l,c,j]
                          for k in K for c in C for l in [1, 2] for j in J)
detention_cost = quicksum(x[k,l,c]*Delta[j]*Y[j][k]*max_expression_detention[k,l,c,j]
                          for k in K for c in C for l in [1, 3] for j in J)
stack_cost = quicksum(x[k,2,c]*(S + S_prime*(s[k,3,c] -
                                             (s[k,2,c] + Phi[k][2]))) for k in K for c

priority_cost = quicksum(x[k,l,c]*rho[k]*(s[k,l,c] - R[k])
                         for k in K for c in C for l in [1, 2])

#Set Objective Function:
model.setObjective(demurrage_cost + detention_cost
                   + stack_cost + priority_cost, GRB.MINIMIZE)

#constraint #1: resource (chassis c) constraints:
model.addConstrs(
    M*(1 - x[k,l,c]) + M*(1 - x[kp,lp,c]) + (s[k,l,c] - s[kp,lp,c]) >=
    Phi[kp][lp] + D[lp][l] - M * z[k, l, kp, lp, c]
    for k in K for l in L for kp in K for lp in L for c in C if
    (not(k == kp and l == lp) and (k <= kp)))

model.addConstrs(
    M*(1 - x[k,l,c]) + M*(1 - x[kp,lp,c]) + (s[kp,lp,c] - s[k,l,c]) >=
    Phi[k][l] + D[l][lp] - M * (1-z[k, l, kp, lp, c])
    for k in K for l in L for kp in K for lp in L for c in C if
    (not(k == kp and l == lp) and (k <= kp)));

#constraint #2: release date constraint:
model.addConstrs(
    s[k,l,c]>= R[k]*x[k,l,c] for k in K for l in L for c in C);

#constraint #3: containers going through leg 2 must go through leg 3:
model.addConstrs(
    quicksum(x[k,2,c] for c in C)==quicksum(x[k,3,c] for c in C) for k in K);

#constraint #4: container must go through leg 1 OR leg 2/leg 3:
```

```python
    model.addConstrs(
        (quicksum(x[k,1,c] for c in C) + quicksum(x[k,2,c] for c in C))==1 for k in K);

    #constraint #5: precedence constraint - leg 2 must be before leg 3:
    model.addConstrs(
        M*(quicksum(x[k,1,c] for c in C)) + quicksum(s[k,3,c] for c in C) >=
        (quicksum(s[k,2,c] for c in C) + Phi[k][2]) for k in K);

    #constraint #6: Bind x_klc = 0 to s_klc = 0:
    model.addConstrs(M*x[k,l,c] >= s[k,l,c] for k in K for l in L for c in C);

    #constraint #7: Domain of start time must be >= 0
    model.addConstrs(s[k,l,c] >=0 for k in K for l in L for c in C);

    model.optimize()

    x_values = {}
    s_values = {}

    for k in K:
        for l in L:
            for c in C:
                x_values[(k,l,c)] = int(x[k,l,c].X)
                if x[k,l,c].X >=0.9:
                    s_values[(k,l,c)] = s[k,l,c].X

    total_dem_cost = int(demurrage_cost.getValue())
    total_det_cost = int(detention_cost.getValue())
    total_stk_cost = int(stack_cost.getValue())
    total_pri_cost = int(priority_cost.getValue())
    total_cost = int(model.ObjVal)
    gap = model.MIPGap*100

    print("\nBEST SOLUTION FOUND:\n")
    print("Total Demurrage Cost: ", total_dem_cost)
    print("Total Detention Cost: ", total_det_cost)
    print("Total Stack Cost: ", total_stk_cost)
    print("Total Priority Penalty Cost: ", total_pri_cost)
    print("Total Overall Cost: ", total_cost)
    print("Gap at Best Solution: (%)", gap)

    return x_values, s_values
```

In [5]:

```python
# Visualize optimized schedule
from datetime import datetime, timedelta
import plotly.graph_objects as go

def plot_schedule(x, s):
    '''
    produce schedule for each container and each chassis over time horizon based on gurobi

    ARGS:
    (dict) x:
    (dict) s:

    RETURN:
    None:     plots only
    '''

    raw_df = []
    start_date = datetime(2020, 11, 1, 0, 0)

    for k in K:
        for l in L:
            for c in C:
```

```
                    if (x[k,l,c] != 0):
                        raw_df.append(dict(Chassis=str(c),
                                           Start=start_date + timedelta(days=int(s[k,l,c]),
                                                                         hours=(s[k,l,c] % 1 *
                                           Finish=start_date + timedelta(days=int(s[k,l,c] + P
                                                                          hours=((s[k,l,c] + Pl
                                           Leg=str(l),
                                           Container=str(k))
                                     )

    df = pd.DataFrame(raw_df)
    fig_chassis = px.timeline(df, x_start="Start", x_end="Finish", y="Chassis", color="Leg
    fig_chassis.show()


    fig_container = px.timeline(df, x_start="Start", x_end="Finish", y="Container", color=

    for k in K:
        fig_container.add_trace(
            go.Scatter(
                x = [start_date + timedelta(days=int(R[k]), hours=(R[k] % 1 * 24)),
                     start_date + timedelta(days=int(R[k]), hours=(R[k] % 1 * 24))],
                y = [k-0.4, k+0.4],
                mode = "lines",
                line = go.scatter.Line(color = "black", width = 1),
                showlegend = False
            )
        )
    fig_container.show()

#fig_chassis.write_html(f"{instance_csv_path}_schedule_.html")
#fig_container.write_html("")
```

Model Results on Test Instances

1. test_instance (base)

- Number of containers (k): 10
- Number of carriers (j): 3
- Number of chassis (c): 5

In [6]:

```
instance_csv_path = "./instances/test_instance.csv"
df = read_instance(instance_csv_path)
df
```

Out[6]:

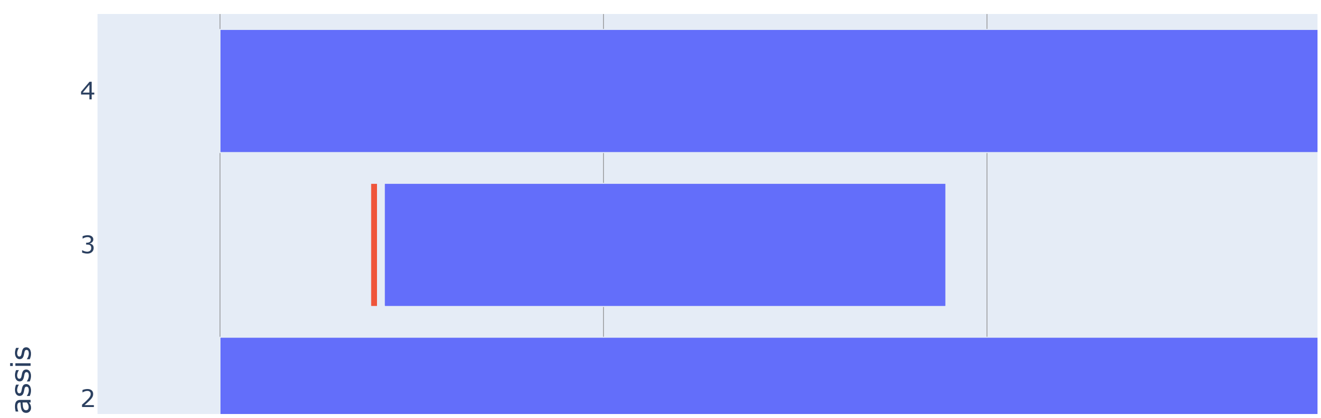
	test_instance	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Ur
header_data	10	3	5	300	15	NaN	NaN	NaN	
container (k)	1	2	3	4	5	6	7	8	
carrier (j)	a	a	a	b	b	b	b	c	
rel_date (R_k)	0	0	0	0	0	4	4	6	
proc_time (P_k)	10	10	20	30	40	1	2	1	
priority (rho_k)	1	2	3	4	5	6	7	8	
free_demurrage (T'_j)	a	3	b	2	c	5	NaN	NaN	

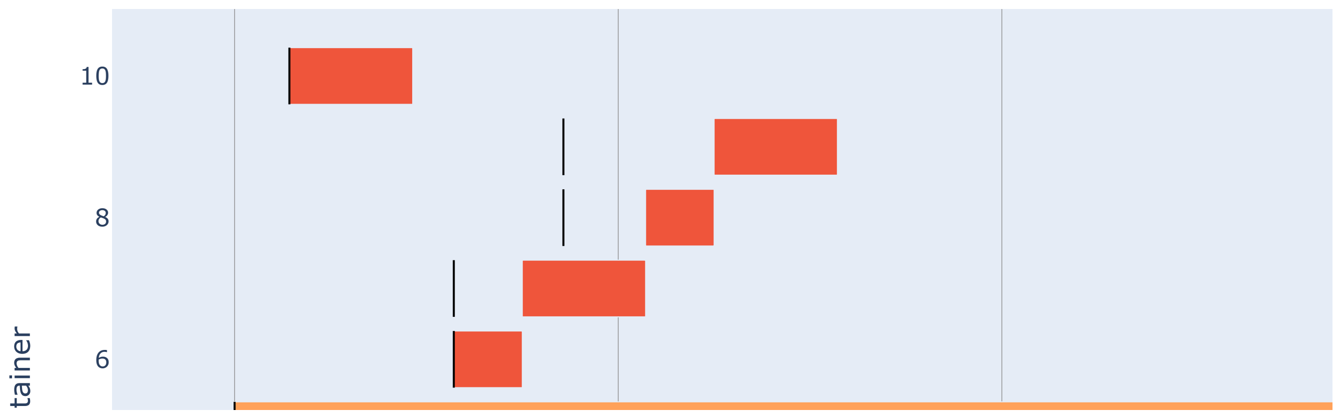
	test_instance	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Ur
demurrage_cost (T_j)	a	1000	b	1400	c	1800	NaN	NaN	
free_detention (delta'_j)	a	21	b	14	c	18	NaN	NaN	
detention_cost (delta_j)	a	50	b	75	c	100	NaN	NaN	
leg_travel_times	leg_1	0.125	leg_2	0.125	leg_3	0.125	NaN	NaN	

In [7]:
```
nK, nJ, nC, S, S_prime, K, J, C, L, R, p, rho, Y, T_prime, T, Delta, Delta_prime, Phi, D,
x, s = run_gurobi_model(S, S_prime, K, J, C, L, R, p, rho, Y, T_prime, T, Delta, Delta_pri
```

```
--------------------------------------------
Warning: your license will expire in 11 days
--------------------------------------------

Using license file /home/sugumarprabhakaran/gurobi.lic
Academic license - for non-commercial use only
Changed value of parameter TimeLimit to 60.0
   Prev: inf  Min: 0.0  Max: inf  Default: inf

BEST SOLUTION FOUND:

Total Demurrage Cost:  0
Total Detention Cost:  3206
Total Stack Cost:  256
Total Priority Penalty Cost:  53
Total Overall Cost:  3517
Gap at Best Solution: (%) 2.0897750293207595
```

In [8]:
```
plot_schedule(x,s)
```





2. 20_instance

- Number of containers (k): 20
- Number of carriers (j): 5
- Number of chassis (c): 5
- Time Horizon: 7 days

In [9]:
```
instance_csv_path = "./instances/20_instance.csv"
df = read_instance(instance_csv_path)
df
```

Out[9]:

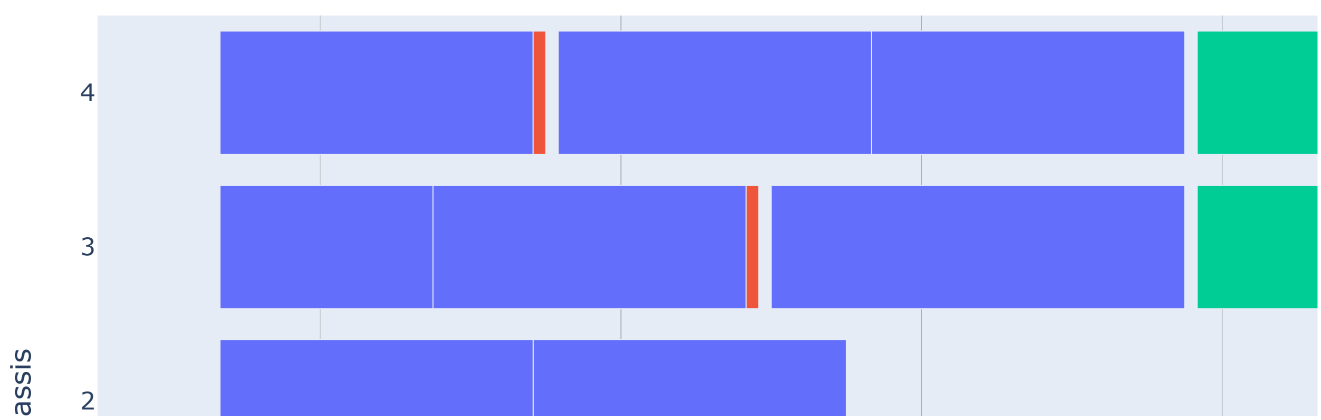
	base_instance -- 20 containers -- 5 carriers -- 25 chassis -- 7-day horizon	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	U
header_data	20.0	5.0	5.0	400.0	15.0	NaN	NaN	NaN	
container (k)	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	
carrier (j)	d	d	d	d	d	d	a	a	
rel_date (R_k)	5	5	5	5	5	5	4	4	
proc_time (P_k)	3	3	3	3	2	3	2	4	

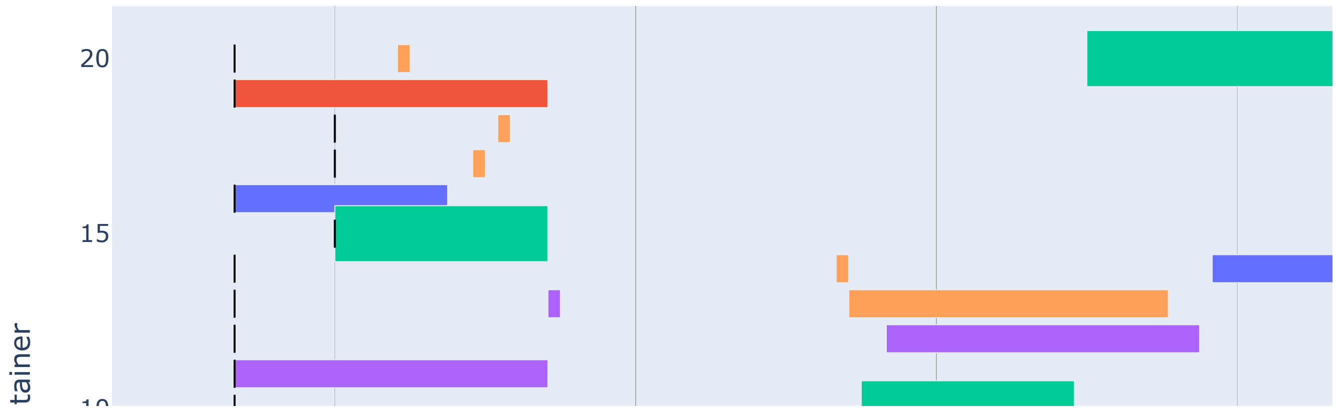
	base_instance -- 20 containers -- 5 carriers -- 25 chassis -- 7-day horizon	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7
priority (rho_k)	19	2	20	9	11	18	7	14
free_demurrage (T'_j)	a	6	b	4	c	2	d	2
demurrage_cost (T_j)	a	100	b	300	c	300	d	150
free_detention (delta'_j)	a	21	b	28	c	28	d	21
detention_cost (delta_j)	a	100	b	50	c	100	d	100
leg_travel_times	leg_1	0.0625	leg_2	0.125	leg_3	0.125	NaN	NaN

In [10]:
```
nK, nJ, nC, S, S_prime, K, J, C, L, R, p, rho, Y, T_prime, T, Delta, Delta_prime, Phi, D,
x, s = run_gurobi_model(S, S_prime, K, J, C, L, R, p, rho, Y, T_prime, T, Delta, Delta_pri
```

```
Changed value of parameter TimeLimit to 60.0
   Prev: inf  Min: 0.0  Max: inf  Default: inf

BEST SOLUTION FOUND:

Total Demurrage Cost:  168
Total Detention Cost:  0
Total Stack Cost:  2329
Total Priority Penalty Cost:  476
Total Overall Cost:  2974
Gap at Best Solution: (%) 99.99999999999993
```

In [11]:
```
plot_schedule(x,s)
```





3. 15_instance

- Number of containers (k): 15
- Number of carriers (j): 5
- Number of chassis (c): 6
- Time Horizon: 30 days

In [12]:
```
instance_csv_path = "./instances/15_instance.csv"
df = read_instance(instance_csv_path)
df
```

Out[12]:

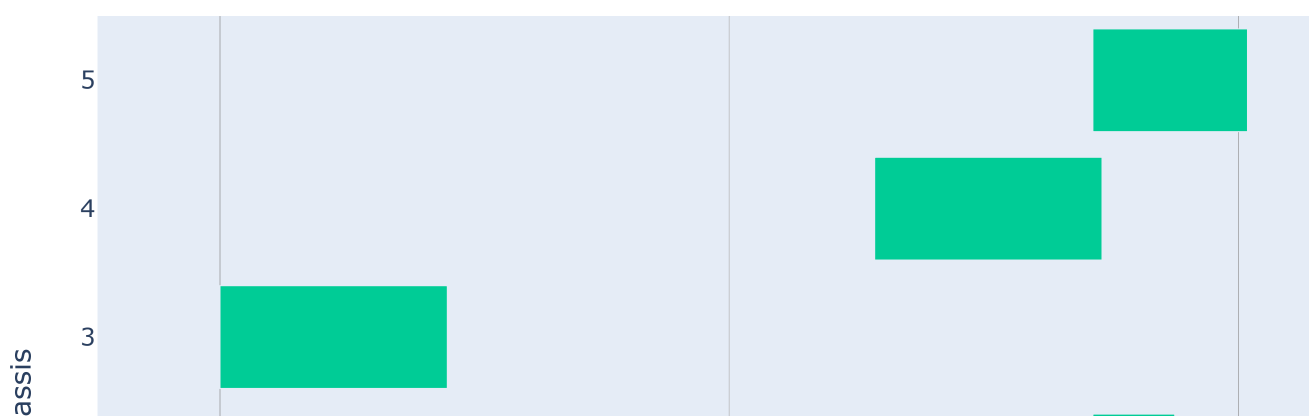
	15_instance -- 15 containers -- 5 carriers -- 6 chassis -- 30-day horizon	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnam
header_data	15.0	5.0	6.0	400	
container (k)	1.0	2.0	3.0	4.0	
carrier (j)	e	a	d	d	
rel_date (R_k)	21	9	18	23	
proc_time (P_k)	2.6731484892945785	3.1512902868403145	2.8166907028228287	2.6630357717677327	2.37014403285
priority (rho_k)	2	7	9	10	

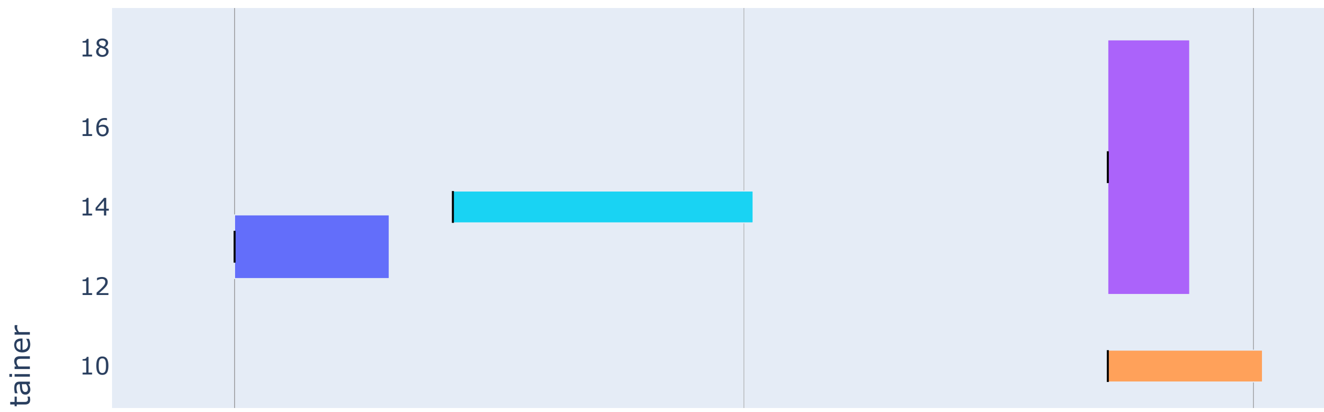
	15_instance -- 15 containers -- 5 carriers -- 6 chassis -- 30-day horizon	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnam
free_demurrage (T'_j)	a	6	b	4	
demurrage_cost (T_j)	a	100	b	300	
free_detention (delta'_j)	a	21	b	28	
detention_cost (delta_j)	a	100	b	50	
leg_travel_times	leg_1	0.0625	leg_2	0.125	

In [13]:
```
nK, nJ, nC, S, S_prime, K, J, C, L, R, p, rho, Y, T_prime, T, Delta, Delta_prime, Phi, D,
x, s = run_gurobi_model(S, S_prime, K, J, C, L, R, p, rho, Y, T_prime, T, Delta, Delta_pri
```

```
Changed value of parameter TimeLimit to 60.0
   Prev: inf  Min: 0.0  Max: inf  Default: inf

BEST SOLUTION FOUND:

Total Demurrage Cost:  0
Total Detention Cost:  0
Total Stack Cost:  -21
Total Priority Penalty Cost:  14
Total Overall Cost:  -7
Gap at Best Solution: (%) 0.0
```

In [14]:
```
plot_schedule(x,s)
```





4. 14_instance

- Number of containers (k): 14
- Number of carriers (j): 3
- Number of chassis (c): 4
- Total time horizon: 10 days

In [18]:
```
instance_csv_path = "./instances/14_instance.csv"
df = read_instance(instance_csv_path)
df
```

Out[18]:

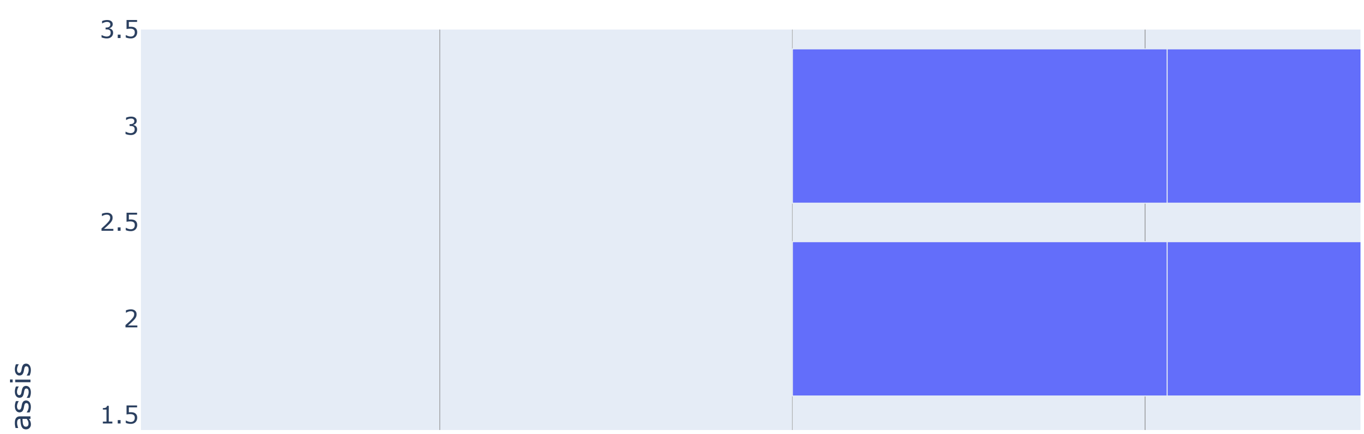
	14_instance -- 14 containers -- 3 carriers -- 4 chassis -- 10-day horizon	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnam
header_data	14.0	3.0	4.0	300	
container (k)	1.0	2.0	3.0	4.0	
carrier (j)	a	b	a	b	
rel_date (R_k)	3	5	3	5	
proc_time (P_k)	2.6731484892945785	3.1512902868403145	2.8166907028228287	2.6630357717677327	2.37014403285
priority (rho_k)	9	7	5	12	
free_demurrage (T'_j)	a	6	b	4	

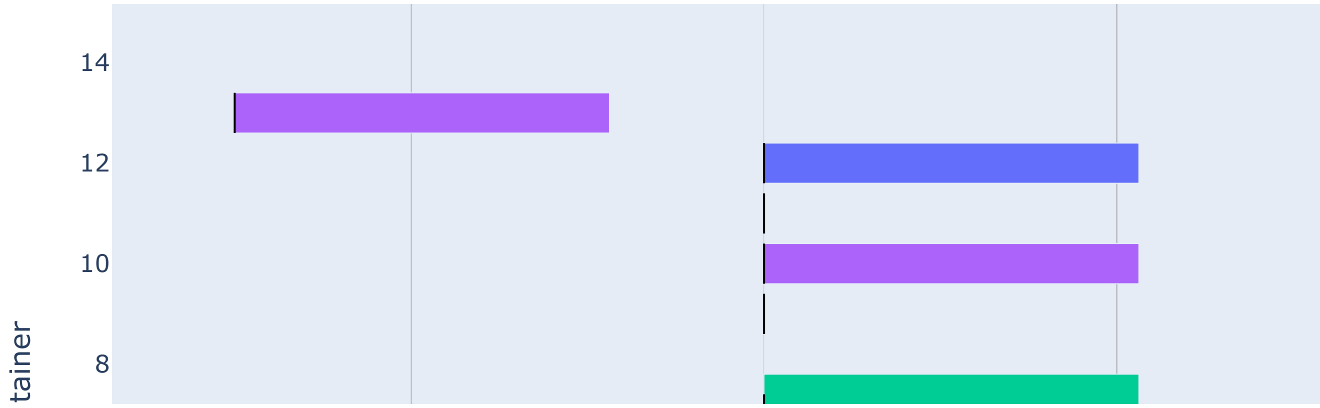
	14_instance -- 14 containers -- 3 carriers -- 4 chassis -- 10-day horizon	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnam
demurrage_cost (T_j)	a	100	b	300	
free_detention (delta'_j)	a	21	b	28	
detention_cost (delta_j)	a	100	b	50	
leg_travel_times	leg_1	0.0625	leg_2	0.125	

In [19]:
```
nK, nJ, nC, S, S_prime, K, J, C, L, R, p, rho, Y, T_prime, T, Delta, Delta_prime, Phi, D,
x, s = run_gurobi_model(S, S_prime, K, J, C, L, R, p, rho, Y, T_prime, T, Delta, Delta_pri
```

```
Changed value of parameter TimeLimit to 60.0
   Prev: inf  Min: 0.0  Max: inf  Default: inf

BEST SOLUTION FOUND:

Total Demurrage Cost:  37
Total Detention Cost:  0
Total Stack Cost:  0
Total Priority Penalty Cost:  104
Total Overall Cost:  141
Gap at Best Solution: (%) 99.7352162400706
```

In [20]:
```
plot_schedule(x,s)
```





5. 12_instance

- Number of containers (k): 12
- Number of carriers (j): 3
- Number of chassis (c): 3
- Total time horizon: 21 days

In [15]:
```
instance_csv_path = "./instances/12_instance.csv"
df = read_instance(instance_csv_path)
df
```

Out[15]:

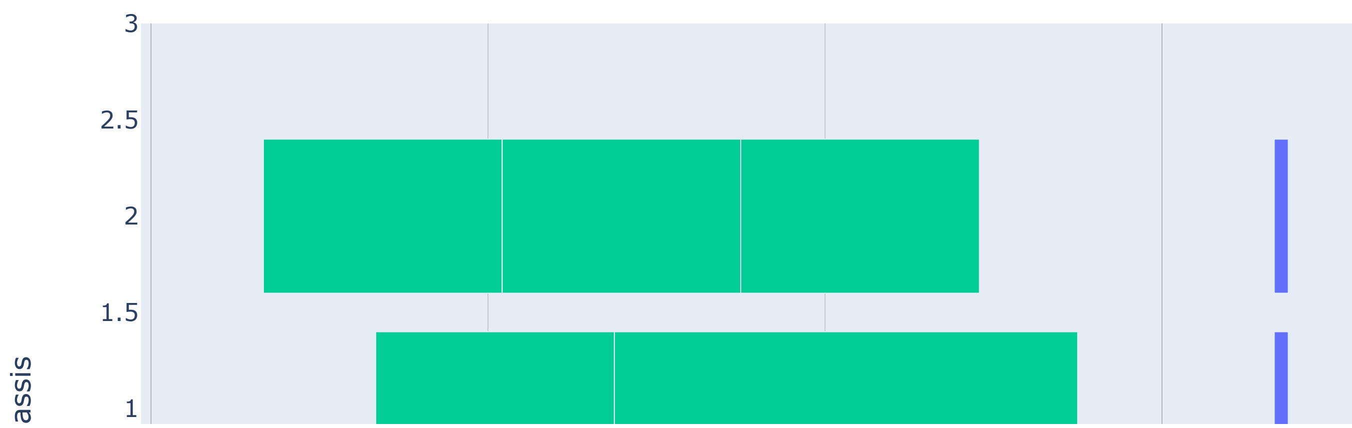
	12_instance -- 12 containers -- 3 carriers -- 3 chassis -- 21-day horizon	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnam
header_data	12.0	3.0	3.0	200	
container (k)	1.0	2.0	3.0	4.0	
carrier (j)	a	b	a	b	
rel_date (R_k)	9	7	7	7	
proc_time (P_k)	2.6731484892945785	3.1512902868403145	2.8166907028228287	2.6630357717677327	2.37014403285
priority (rho_k)	7	12	5	11	
free_demurrage (T'_j)	a	6	b	4	
demurrage_cost (T_j)	a	100	b	300	

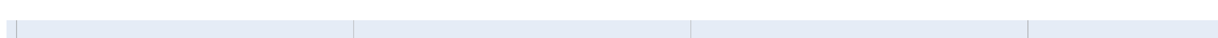
	12_instance -- 12 containers -- 3 carriers -- 3 chassis -- 21-day horizon	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnam
free_detention (delta'_j)	a	21	b	28	
detention_cost (delta_j)	a	100	b	50	
leg_travel_times	leg_1	0.0625	leg_2	0.125	

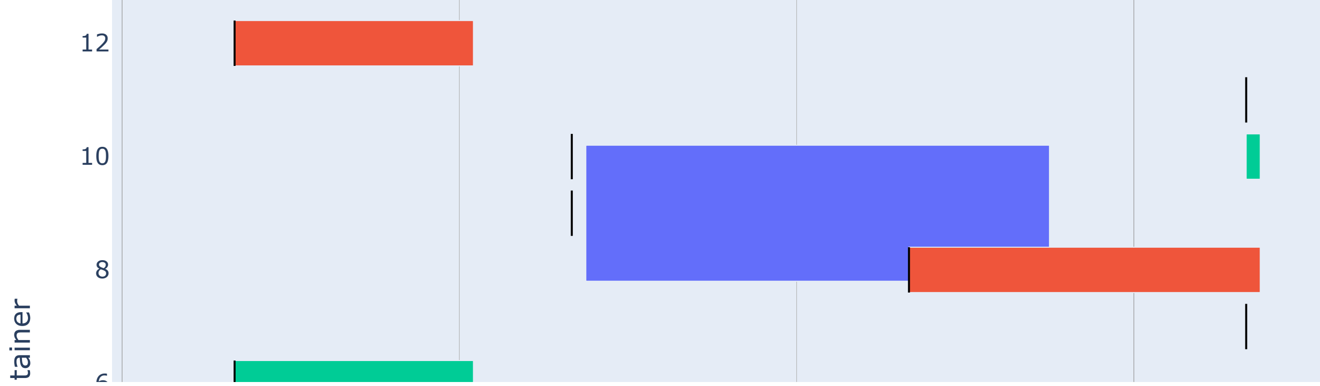
In [16]:
```
nK, nJ, nC, S, S_prime, K, J, C, L, R, p, rho, Y, T_prime, T, Delta, Delta_prime, Phi, D,
x, s = run_gurobi_model(S, S_prime, K, J, C, L, R, p, rho, Y, T_prime, T, Delta, Delta_pri
```

```
Changed value of parameter TimeLimit to 60.0
   Prev: inf  Min: 0.0  Max: inf  Default: inf

BEST SOLUTION FOUND:

Total Demurrage Cost:  0
Total Detention Cost:  0
Total Stack Cost:  -287
Total Priority Penalty Cost:  114
Total Overall Cost:  -173
Gap at Best Solution: (%) 0.0
```

In [17]:
```
plot_schedule(x,s)
```






12
10
8
6
tainer