

# Railway Single-Track Sequencing

## A big-M MILP, in every modelling dialect

A small, complete, self-contained mixed-integer linear program from railway operations. Four trains share one single-track segment (a tunnel / bridge / passing-loop bottleneck): at most one train may occupy it at a time and a minimum headway must elapse between trains. Decide each train's entry time so as to minimise the total weighted clearance time. The mutual exclusion of any two trains is a disjunction, linearised with the classic **big-M** technique.

The same model appears on the following pages as a complete problem statement, the mathematical model, LaTeX, markdown/mathcal, and ~22 executable encodings — one representation per page.

Optimal objective:  $\sum_i w_i \cdot C_i = 66$  (order  $B \rightarrow D \rightarrow C \rightarrow A$ ) — verified with CBC, GLPK and OR-Tools CP-SAT.

# 1 · Complete problem description

sets, parameters, variables, data, optimum

**Sets.**  $N = \{A, B, C, D\}$  (trains);  $P = \{(i, j) : i < j\}$  (ordered index pairs).

**Parameters.**  $r_i$  release (earliest entry);  $p_i$  running time on the segment;  $w_i$  priority weight;  $h$  minimum headway ( $= 1$ ); big-M constant  $M = \max_i r_i + \sum_i p_i + (|N|-1) \cdot h = 4 + 14 + 3 = 21$ .

**Variables.**  $t_i \geq 0$  entry time;  $C_i \geq 0$  clearance time;  $y_{ij} \in \{0, 1\}$  with  $y_{ij} = 1$  iff train  $i$  enters before train  $j$ .

Data instance

| train | release $r_i$ | running $p_i$ | weight $w_i$ |
|-------|---------------|---------------|--------------|
| A     | 0             | 5             | 1            |
| B     | 2             | 3             | 2            |
| C     | 1             | 4             | 1            |
| D     | 4             | 2             | 3            |

Optimal schedule

| train | enter $t_i$ | clear $C_i$ | $w_i \cdot C_i$ |
|-------|-------------|-------------|-----------------|
| B     | 2           | 5           | 10              |
| D     | 6           | 8           | 24              |
| C     | 9           | 13          | 13              |
| A     | 14          | 19          | 19              |
|       |             | total       | 66              |

## 2 · The mathematical model

the MILP, solver-independent

---

```
minimize    sum_{i in N} w_i * C_i                (weighted clearance)

subject to
  (release)   t_i >= r_i                          for all i in N
  (clearance) C_i = t_i + p_i                      for all i in N
  (i before j) t_j >= t_i + p_i + h - M*(1 - y_ij) for all (i,j) in P
  (j before i) t_i >= t_j + p_j + h - M*y_ij       for all (i,j) in P
  (domains)   t_i >= 0, C_i >= 0, y_ij in {0,1}
```

### Why big-M works

For a pair (i,j) the binary  $y_{ij}$  selects which of the two mutually exclusive sequencing constraints is enforced:

- \*  $y_{ij} = 1$  : "i before j" is tight ( $t_j \geq t_i + p_i + h$ ),  
"j before i" becomes  $t_i \geq t_j + p_j + h - M$  (vacuous).
- \*  $y_{ij} = 0$  : symmetric -- the other constraint binds.

M must be large enough never to cut a feasible schedule, yet as small as possible so the LP relaxation stays tight.  $M = 21$  is exactly tight here.

### 3 · LaTeX

problem.tex — standalone, compile with pdflatex

```
% Railway single-track sequencing -- a big-M MILP.
% Standalone document: compile with pdflatex problem.tex
\documentclass[11pt]{article}
\usepackage[a4paper,margin=2.4cm]{geometry}
\usepackage{amsmath,amssymb}
\usepackage{booktabs}
\usepackage{array}

\title{Railway Single-Track Sequencing\[\[2pt\]\large A big-M MILP}
\author{lp2graph example catalogue}
\date{}

\begin{document}
\maketitle

\section*{Problem}

A set of trains must each traverse one shared \emph{single-track segment}
(a tunnel, bridge, or passing-loop bottleneck). At most one train may
occupy the segment at a time, and a minimum \emph{headway}  $h$  must elapse
between one train clearing the segment and the next entering it. Each train
 $i$  has an earliest entry (release) time  $r_i$ , a running time  $p_i$ , and a
priority weight  $w_i$ . The dispatcher chooses each entry time  $t_i$  (and
hence the order) to minimise the total weighted clearance time, where the
clearance time is  $C_i = t_i + p_i$ .

The mutual exclusion of any two trains  $i, j$  -- \emph{either  $i$  before  $j$ 
or  $j$  before  $i$ } -- is a disjunction, linearised with a binary ordering
variable and the \textbf{big-M} technique.

\section*{Sets, parameters, variables}

\begin{align*}
\mathcal{N} &= \{A, B, C, D\} \text{ \& \text{trains}} \\
\mathcal{P} &= \{(i, j) \in \mathcal{N} \times \mathcal{N} : i \prec j\} \text{ \& \text{ordered index pairs}} \\
r_i, p_i, w_i &\text{ \& \text{release, running time, weight of train } } i \\
h &\text{ \& \text{minimum headway}} \\
M &= \max_i r_i + \textstyle\sum_{i \in \mathcal{N}} p_i + (|\mathcal{N}|-1)h \text{ \& \text{big-M constant}} \\
t_i &\geq 0 \text{ \& \text{entry time of train } } i \\
C_i &\geq 0 \text{ \& \text{clearance (completion) time of train } } i \\
y_{ij} &\in \{0, 1\} \text{ \& } y_{ij} = 1 \text{ \& \text{iff \text{train } } i \text{ enters before train } } j, (i, j) \in \mathcal{P}
\end{align*}

\section*{Model}

\begin{align}
\min_{t, C, y} \quad & \sum_{i \in \mathcal{N}} w_i C_i \\
\text{s.t.} \quad & t_i \geq r_i \text{ \& \forall } i \in \mathcal{N} \\
& C_i = t_i + p_i \text{ \& \forall } i \in \mathcal{N} \\
& t_j \geq t_i + p_i + h - M y_{ij} \text{ \& \forall } (i, j) \in \mathcal{P} \\
& t_i \geq t_j + p_j + h - M y_{ij} \text{ \& \forall } (i, j) \in \mathcal{P} \\
& t_i \geq 0, C_i \geq 0, y_{ij} \in \{0, 1\}.
\end{align}

\paragraph{why big-M works.} For a pair  $(i, j)$  the binary  $y_{ij}$  selects
which of the two mutually exclusive sequencing constraints is active. If
 $y_{ij} = 1$ , constraint (4) becomes the tight  $t_j \geq t_i + p_i + h$  while
constraint (5) reads  $t_i \geq t_j + p_j + h - M$ , which for large  $M$  is
vacuous. If  $y_{ij} = 0$  the roles swap.  $M$  must be large enough never to cut
a feasible schedule, yet as small as possible so the LP relaxation stays
tight.

\section*{Data instance}

\begin{center}
\begin{tabular}{crrr}
\toprule
train &  $r_i$  &  $p_i$  &  $w_i$  \\
\midrule
A & 0 & 5 & 1 \\
B & 2 & 3 & 2 \\
C & 1 & 4 & 1 \\
D & 4 & 2 & 3 \\
\bottomrule
\end{tabular}
\end{center}

\noindent Headway  $h = 1$ , and
 $M = \max_i r_i + \sum_i p_i + (|\mathcal{N}|-1)h = 4 + 14 + 3 = 21$ .

\section*{Optimal solution}

The optimal weighted clearance is  $\sum_i w_i C_i = 66$ , attained by the
order  $B \rightarrow D \rightarrow C \rightarrow A$ :

\begin{center}
\begin{tabular}{crrr}
\toprule
train &  $t_i$  &  $C_i$  &  $w_i C_i$  \\
\midrule
B & 2 & 5 & 10 \\
D & 6 & 8 & 24 \\
C & 9 & 13 & 13 \\
A & 14 & 19 & 19 \\
\midrule
\multicolumn{3}{r}{total} & \textbf{66} \\
\bottomrule
\end{tabular}
\end{center}

\end{document}
```

## 4 · Markdown / mathcal

README.md model section (GitHub/MathJax)

```
## 2. The mathematical model (markdown / ``mathcal``)

> Rendered by GitHub/MathJax. The identical model in standalone LaTeX is in
> [``problem.tex``](problem.tex).

**Sets**


$$\mathcal{N} = \{A, B, C, D\} \quad \text{(trains)}, \quad \mathcal{P} = \{(i, j) \in \mathcal{N} \times \mathcal{N} : i \prec j\} \quad \text{(ordered index pairs)}$$


**Parameters**


$$r_i \quad \text{(release)}, \quad p_i \quad \text{(running time)}, \quad w_i \quad \text{(weight)}, \quad h \quad \text{(headway)}, \quad M = \max_i r_i + \sum_{i \in \mathcal{N}} p_i + (|\mathcal{N}| - 1)h$$


**Decision variables**


$$t_i \geq 0 \quad \text{(entry time)}, \quad c_i \geq 0 \quad \text{(clearance time)}, \quad y_{ij} \in \{0, 1\} \quad \forall (i, j) \in \mathcal{P}$$


with the reading  $y_{ij}=1$  iff train  $i$  enters the segment before train  $j$ .

**Model**


$$\begin{aligned} & \min_{t, c, y} \quad \sum_{i \in \mathcal{N}} w_i c_i \quad \text{(weighted clearance)} \\ & \text{s.t.} \quad & t_i \geq r_i \quad \forall i \in \mathcal{N} \quad \text{(release)} \\ & & c_i = t_i + p_i \quad \forall i \in \mathcal{N} \quad \text{(clearance)} \\ & & t_j \geq t_i + p_i + h - M(1 - y_{ij}) \quad \forall (i, j) \in \mathcal{P} \quad \text{($i$ before $j$)} \\ & & t_i \geq t_j + p_j + h - M y_{ij} \quad \forall (i, j) \in \mathcal{P} \quad \text{($j$ before $i$)} \\ & & t_i \geq 0, \quad c_i \geq 0, \quad y_{ij} \in \{0, 1\}. \end{aligned}$$


**Why big-M works.** For a pair  $(i, j)$  the binary  $y_{ij}$  *selects* which of the two mutually-exclusive sequencing constraints is enforced:



- If  $y_{ij}=1$ : the first constraint becomes the tight  $t_j \geq t_i + p_i + h$  (train  $i$  then  $j$ ), while the second becomes  $t_i \geq t_j + p_j + h - M$  – with  $M$  large this right-hand side is so negative that the constraint is vacuous.
- If  $y_{ij}=0$ : symmetric – the second constraint binds and the first is switched off.

 $M$  must be large enough never to wrongly cut a feasible schedule, but no larger, because an oversized  $M$  weakens the LP relaxation and slows the branch-and-bound. The bound above is exactly tight enough.
```

## 5 · PuLP

model\_pulp.py — Python

```
"""Railway single-track sequencing (big-M MILP) -- PuLP.

    pip install pulp
    python model_pulp.py

Also exports model.lp (CPLEX LP) and model.mps so the portable-format files
in this folder are guaranteed to be the exact same model.
"""
import itertools
import pulp

# ----- data -----
TRAINS = ["A", "B", "C", "D"]
r = {"A": 0, "B": 2, "C": 1, "D": 4} # release (earliest entry)
p = {"A": 5, "B": 3, "C": 4, "D": 2} # running time on the segment
w = {"A": 1, "B": 2, "C": 1, "D": 3} # priority weight
h = 1 # minimum headway
n = len(TRAINS)
M = max(r.values()) + sum(p.values()) + (n - 1) * h # = 21, valid big-M
PAIRS = list(itertools.combinations(TRAINS, 2)) # ordered index pairs i<j

# ----- model -----
m = pulp.LpProblem("railway_bigm", pulp.LpMinimize)

t = {i: pulp.LpVariable(f"t_{i}", lowBound=0) for i in TRAINS}
C = {i: pulp.LpVariable(f"C_{i}", lowBound=0) for i in TRAINS}
y = {(i, j): pulp.LpVariable(f"y_{i}_{j}", cat="Binary") for (i, j) in PAIRS}

m += pulp.lpSum(w[i] * C[i] for i in TRAINS), "weighted_clearance"

for i in TRAINS:
    m += t[i] >= r[i], f"release_{i}"
    m += C[i] == t[i] + p[i], f"completion_{i}"

for (i, j) in PAIRS:
    m += t[j] >= t[i] + p[i] + h - M * (1 - y[(i, j)]), f"seq_{i}_before_{j}"
    m += t[i] >= t[j] + p[j] + h - M * y[(i, j)], f"seq_{j}_before_{i}"

# ----- solve -----
m.solve(pulp.PULP_CBC_CMD(msg=False))

print("status :", pulp.LpStatus[m.status])
print("objective:", pulp.value(m.objective))
for i in sorted(TRAINS, key=lambda k: pulp.value(t[k])):
    print(f" {i}: enter t={pulp.value(t[i]):.0f} clear C={pulp.value(C[i]):.0f}")

# ----- export portable formats -----
m.writeLP("model.lp")
m.writeMPS("model.mps")
```

## 6 · Gurobi (gurobipy)

model\_gurobi.py — Python

```
"""Railway single-track sequencing (big-M MILP) -- gurobipy.

    pip install gurobipy          # needs a Gurobi licence
    python model_gurobi.py
"""
import itertools
import gurobipy as gp
from gurobipy import GRB

# ----- data -----
TRAINS = ["A", "B", "C", "D"]
r = {"A": 0, "B": 2, "C": 1, "D": 4}
p = {"A": 5, "B": 3, "C": 4, "D": 2}
w = {"A": 1, "B": 2, "C": 1, "D": 3}
h = 1
n = len(TRAINS)
M = max(r.values()) + sum(p.values()) + (n - 1) * h  # = 21
PAIRS = list(itertools.combinations(TRAINS, 2))

# ----- model -----
m = gp.Model("railway_bigm")

t = m.addVars(TRAINS, lb=0.0, name="t")
C = m.addVars(TRAINS, lb=0.0, name="C")
y = m.addVars(PAIRS, vtype=GRB.BINARY, name="y")

m.setObjective(gp.quicksum(w[i] * C[i] for i in TRAINS), GRB.MINIMIZE)

m.addConstrs((t[i] >= r[i] for i in TRAINS), name="release")
m.addConstrs((C[i] == t[i] + p[i] for i in TRAINS), name="completion")
for (i, j) in PAIRS:
    m.addConstr(t[j] >= t[i] + p[i] + h - M * (1 - y[i, j]), name=f"seq_{i}_before_{j}")
    m.addConstr(t[i] >= t[j] + p[j] + h - M * y[i, j], name=f"seq_{j}_before_{i}")

# ----- solve -----
m.optimize()

if m.status == GRB.OPTIMAL:
    print("objective:", m.objVal)
    for i in sorted(TRAINS, key=lambda k: t[k].X):
        print(f" {i}: enter t={t[i].X:.0f} clear C={C[i].X:.0f}")
```

## 7 · CPLEX (DOcplex)

model\_docplex.py — Python

---

```
"""Railway single-track sequencing (big-M MILP) -- CPLEX via DOcplex (Python).

    pip install docplex cplex
    python model_docplex.py

`docplex` is IBM's modelling layer; `cplex` provides the engine (the pip
Community Edition solves models up to 1000 vars/constraints -- plenty here).
"""
import itertools
from docplex.mp.model import Model

# ----- data -----
TRAINS = ["A", "B", "C", "D"]
r = {"A": 0, "B": 2, "C": 1, "D": 4}
p = {"A": 5, "B": 3, "C": 4, "D": 2}
w = {"A": 1, "B": 2, "C": 1, "D": 3}
h = 1
n = len(TRAINS)
M = max(r.values()) + sum(p.values()) + (n - 1) * h  # = 21
PAIRS = list(itertools.combinations(TRAINS, 2))

# ----- model -----
mdl = Model(name="railway_bigm")

t = mdl.continuous_var_dict(TRAINS, lb=0, name="t")
C = mdl.continuous_var_dict(TRAINS, lb=0, name="C")
y = mdl.binary_var_dict(PAIRS, name="y")

mdl.minimize(mdl.sum(w[i] * C[i] for i in TRAINS))

for i in TRAINS:
    mdl.add_constraint(t[i] >= r[i], ctname=f"release_{i}")
    mdl.add_constraint(C[i] == t[i] + p[i], ctname=f"completion_{i}")
for (i, j) in PAIRS:
    mdl.add_constraint(t[j] >= t[i] + p[i] + h - M * (1 - y[(i, j)]), ctname=f"seq_{i}_before_{j}")
    mdl.add_constraint(t[i] >= t[j] + p[j] + h - M * y[(i, j)], ctname=f"seq_{j}_before_{i}")

# ----- solve -----
mdl.solve()

print("objective:", mdl.objective_value)
for i in sorted(TRAINS, key=lambda k: t[k].solution_value):
    print(f" {i}: enter t={t[i].solution_value:.0f} clear C={C[i].solution_value:.0f}")
```



## 8 · Pyomo

model\_pyomo.py — Python

```
"""Railway single-track sequencing (big-M MILP) -- Pyomo.

    pip install pyomo
    python model_pyomo.py          # uses any installed MILP solver (glpk/cbc/gurobi)
"""
import itertools
import pyomo.environ as pyo

# ----- data -----
TRAINS = ["A", "B", "C", "D"]
r = {"A": 0, "B": 2, "C": 1, "D": 4}
p = {"A": 5, "B": 3, "C": 4, "D": 2}
w = {"A": 1, "B": 2, "C": 1, "D": 3}
h = 1
n = len(TRAINS)
M = max(r.values()) + sum(p.values()) + (n - 1) * h    # = 21
PAIRS = list(itertools.combinations(TRAINS, 2))

# ----- model -----
m = pyo.ConcreteModel("railway_bigm")
m.I = pyo.Set(initialize=TRAINS)
m.P = pyo.Set(initialize=PAIRS, dimen=2)

m.t = pyo.Var(m.I, domain=pyo.NonNegativeReals)
m.C = pyo.Var(m.I, domain=pyo.NonNegativeReals)
m.y = pyo.Var(m.P, domain=pyo.Binary)

m.obj = pyo.Objective(expr=sum(w[i] * m.C[i] for i in m.I), sense=pyo.minimize)

m.release = pyo.Constraint(m.I, rule=lambda m, i: m.t[i] >= r[i])
m.completion = pyo.Constraint(m.I, rule=lambda m, i: m.C[i] == m.t[i] + p[i])
m.seq_ij = pyo.Constraint(m.P, rule=lambda m, i, j:
    m.t[j] >= m.t[i] + p[i] + h - M * (1 - m.y[i, j]))
m.seq_ji = pyo.Constraint(m.P, rule=lambda m, i, j:
    m.t[i] >= m.t[j] + p[j] + h - M * m.y[i, j])

# ----- solve -----
solver = pyo.SolverFactory("glpk")    # or "cbc", "gurobi", ...
solver.solve(m)

print("objective:", pyo.value(m.obj))
for i in sorted(TRAINS, key=lambda k: pyo.value(m.t[k])):
    print(f" {i}: enter t={pyo.value(m.t[i]):.0f}  clear C={pyo.value(m.C[i]):.0f}")
```

## 9 · Google OR-Tools

model\_ortools.py — Python (MPSolver)

---

```
"""Railway single-track sequencing (big-M MILP) -- Google OR-Tools (MPSolver).

    pip install ortools
    python model_ortools.py

Uses the linear MILP wrapper (CBC backend) so the big-M formulation is kept
verbatim. (OR-Tools' CP-SAT could model the disjunction natively without a
big-M, but the point here is to show the classic big-M MILP.)
"""
import itertools
from ortools.linear_solver import pywraplp

# ----- data -----
TRAINS = ["A", "B", "C", "D"]
r = {"A": 0, "B": 2, "C": 1, "D": 4}
p = {"A": 5, "B": 3, "C": 4, "D": 2}
w = {"A": 1, "B": 2, "C": 1, "D": 3}
h = 1
n = len(TRAINS)
M = max(r.values()) + sum(p.values()) + (n - 1) * h # = 21
PAIRS = list(itertools.combinations(TRAINS, 2))

# ----- model -----
solver = pywraplp.Solver.CreateSolver("CBC")
INF = solver.infinity()

t = {i: solver.NumVar(0.0, INF, f"t_{i}") for i in TRAINS}
C = {i: solver.NumVar(0.0, INF, f"C_{i}") for i in TRAINS}
y = {(i, j): solver.BoolVar(f"y_{i}_{j}") for (i, j) in PAIRS}

for i in TRAINS:
    solver.Add(t[i] >= r[i])
    solver.Add(C[i] == t[i] + p[i])
for (i, j) in PAIRS:
    solver.Add(t[j] >= t[i] + p[i] + h - M * (1 - y[(i, j)]))
    solver.Add(t[i] >= t[j] + p[j] + h - M * y[(i, j)])

solver.Minimize(solver.Sum(w[i] * C[i] for i in TRAINS))

# ----- solve -----
status = solver.Solve()
if status == pywraplp.Solver.OPTIMAL:
    print("objective:", solver.Objective().Value())
    for i in sorted(TRAINS, key=lambda k: t[k].solution_value()):
        print(f" {i}: enter t={t[i].solution_value():.0f} clear C={C[i].solution_value():.0f}")
```

## 10 · Python-MIP

model\_python\_mip.py — Python

```
"""Railway single-track sequencing (big-M MILP) -- Python-MIP.

    pip install mip
    python model_python_mip.py
"""
import itertools
from mip import Model, xsum, minimize, BINARY, CONTINUOUS, OptimizationStatus

# ----- data -----
TRAINS = ["A", "B", "C", "D"]
r = {"A": 0, "B": 2, "C": 1, "D": 4}
p = {"A": 5, "B": 3, "C": 4, "D": 2}
w = {"A": 1, "B": 2, "C": 1, "D": 3}
h = 1
n = len(TRAINS)
M = max(r.values()) + sum(p.values()) + (n - 1) * h  # = 21
PAIRS = list(itertools.combinations(TRAINS, 2))

# ----- model -----
m = Model("railway_bigm")

t = {i: m.add_var(name=f"t_{i}", lb=0.0, var_type=CONTINUOUS) for i in TRAINS}
C = {i: m.add_var(name=f"C_{i}", lb=0.0, var_type=CONTINUOUS) for i in TRAINS}
y = {(i, j): m.add_var(name=f"y_{i}_{j}", var_type=BINARY) for (i, j) in PAIRS}

m.objective = minimize(xsum(w[i] * C[i] for i in TRAINS))

for i in TRAINS:
    m += t[i] >= r[i]
    m += C[i] == t[i] + p[i]
for (i, j) in PAIRS:
    m += t[j] >= t[i] + p[i] + h - M * (1 - y[(i, j)])
    m += t[i] >= t[j] + p[j] + h - M * y[(i, j)]

# ----- solve -----
status = m.optimize()
if status in (OptimizationStatus.OPTIMAL, OptimizationStatus.FEASIBLE):
    print("objective:", m.objective_value)
    for i in sorted(TRAINS, key=lambda k: t[k].x):
        print(f" {i}: enter t={t[i].x:.0f} clear C={C[i].x:.0f}")
```

## 11 · CVXPY

model\_cvxpy.py — Python

---

```
"""Railway single-track sequencing (big-M MILP) -- CVXPY.

    pip install cvxpy
    python model_cvxpy.py          # needs a MILP-capable backend (GLPK_MI/CBC/SCIP/...)

CVXPY is matrix/vector oriented, so trains are indexed 0..n-1.
"""
import itertools
import numpy as np
import cvxpy as cp

# ----- data (index order A,B,C,D) -----
TRAINS = ["A", "B", "C", "D"]
r = np.array([0, 2, 1, 4])
p = np.array([5, 3, 4, 2])
w = np.array([1, 2, 1, 3])
h = 1
n = len(TRAINS)
M = int(r.max() + p.sum() + (n - 1) * h)  # = 21
PAIRS = list(itertools.combinations(range(n), 2))

# ----- model -----
t = cp.Variable(n, nonneg=True, name="t")
C = cp.Variable(n, nonneg=True, name="C")
y = cp.Variable(len(PAIRS), boolean=True, name="y")

constraints = [t >= r, C == t + p]
for k, (i, j) in enumerate(PAIRS):
    constraints += [
        t[j] >= t[i] + p[i] + h - M * (1 - y[k]),
        t[i] >= t[j] + p[j] + h - M * y[k],
    ]

prob = cp.Problem(cp.Minimize(w @ C), constraints)

# ----- solve -----
prob.solve(solver=cp.GLPK_MI)          # or cp.CBC, cp.SCIP, ...

print("objective:", prob.value)
order = sorted(range(n), key=lambda i: t.value[i])
for i in order:
    print(f" {TRAINS[i]}: enter t={t.value[i]:.0f}  clear C={C.value[i]:.0f}")
```

## 12 · JuMP

model\_jump.jl — Julia

```
# Railway single-track sequencing (big-M MILP) -- JuMP (Julia).
# import Pkg; Pkg.add(["JuMP", "HiGHS"])
# julia model_jump.jl
using JuMP, HiGHS

trains = [:A, :B, :C, :D]
r = Dict{:A => 0, :B => 2, :C => 1, :D => 4} # release
p = Dict{:A => 5, :B => 3, :C => 4, :D => 2} # running time
w = Dict{:A => 1, :B => 2, :C => 1, :D => 3} # weight
h = 1 # headway
n = length(trains)
M = maximum(values(r)) + sum(values(p)) + (n - 1) * h # big-M = 21
pairs = [(trains[a], trains[b]) for a in 1:n for b in (a + 1):n]

model = Model(HiGHS.Optimizer)
@variable(model, t[trains] >= 0) # entry time
@variable(model, C[trains] >= 0) # clearance time
@variable(model, y[pairs], Bin) # y[(i,j)]=1 iff i before j

@objective(model, Min, sum(w[i] * C[i] for i in trains))
@constraint(model, [i in trains], t[i] >= r[i])
@constraint(model, [i in trains], C[i] == t[i] + p[i])
@constraint(model, [(i, j) in pairs], t[j] >= t[i] + p[i] + h - M * (1 - y[(i, j)]))
@constraint(model, [(i, j) in pairs], t[i] >= t[j] + p[j] + h - M * y[(i, j)])

optimize!(model)

println("objective = ", objective_value(model))
for i in sort(trains, by = i -> value(t[i]))
    println(" $i: enter t=", round{Int, value(t[i])}, " clear C=", round{Int, value(C[i])})
end
```

## 13 · ompr / ROI

model\_ompr.R — R

```
# Railway single-track sequencing (big-M MILP) -- ompr (R).
#   install.packages(c("ompr", "ompr.roi", "ROI.plugin.glpk", "magrittr"))
#   Rscript model_ompr.R
library(ompr)
library(ompr.roi)
library(ROI.plugin.glpk)
library(magrittr)

trains <- c("A", "B", "C", "D")
n <- length(trains)
r <- c(0, 2, 1, 4) # release      (positions match `trains`)
p <- c(5, 3, 4, 2) # running time
w <- c(1, 2, 1, 3) # weight
h <- 1            # headway
M <- max(r) + sum(p) + (n - 1) * h # big-M = 21

model <- MIPModel() %>%
  add_variable(t[i], i = 1:n, type = "continuous", lb = 0) %>%
  add_variable(C[i], i = 1:n, type = "continuous", lb = 0) %>%
  add_variable(y[i, j], i = 1:n, j = 1:n, i < j, type = "binary") %>%
  set_objective(sum_over(w[i] * C[i], i = 1:n), "min") %>%
  add_constraint(t[i] >= r[i], i = 1:n) %>%
  add_constraint(C[i] == t[i] + p[i], i = 1:n) %>%
  add_constraint(t[j] >= t[i] + p[i] + h - M * (1 - y[i, j]), i = 1:n, j = 1:n, i < j) %>%
  add_constraint(t[i] >= t[j] + p[j] + h - M * y[i, j], i = 1:n, j = 1:n, i < j)

res <- solve_model(model, with_ROI(solver = "glpk"))

cat("objective =", objective_value(res), "\n")
tv <- get_solution(res, t[i])
Cv <- get_solution(res, C[i])
for (i in order(tv$value)) {
  cat(sprintf(" %s: enter t=%g clear C=%g\n", trains[i], tv$value[i], Cv$value[i]))
}
```

## 14 · intlinprog

model\_matlab.m — MATLAB / Octave

```
% Railway single-track sequencing (big-M MILP) -- MATLAB (intlinprog).
% run in MATLAB: model_matlab
% (Octave users: replace the intlinprog call with the bundled `glpk`.)
%
% Variable vector x = [ t(1..n) , C(1..n) , y(1..np) ], pairs i<j.

trains = {'A','B','C','D'};
r = [0 2 1 4]; % release (earliest entry)
p = [5 3 4 2]; % running time on the segment
w = [1 2 1 3]; % priority weight
h = 1; % minimum headway
n = numel(trains);
M = max(r) + sum(p) + (n - 1) * h; % big-M = 21

pairs = nchoosek(1:n, 2); % rows [i j] with i<j
np = size(pairs, 1);
N = 2 * n + np; % total variables
it = @(i) i; % index of t_i
iC = @(i) n + i; % index of C_i
iy = @(k) 2 * n + k; % index of y_k

% --- objective: min sum_i w_i C_i ---
f = zeros(N, 1);
f(iC(1:n)) = w;

% --- bounds + integrality (release t_i >= r_i as a lower bound) ---
lb = zeros(N, 1); lb(it(1:n)) = r;
ub = inf(N, 1); ub(iy(1:np)) = 1;
intcon = iy(1:np);

% --- equalities: C_i - t_i = p_i ---
Aeq = zeros(n, N); beq = zeros(n, 1);
for i = 1:n
    Aeq(i, iC(i)) = 1;
    Aeq(i, it(i)) = -1;
    beq(i) = p(i);
end

% --- big-M inequalities (A x <= b) ---
A = zeros(2 * np, N); b = zeros(2 * np, 1);
row = 0;
for k = 1:np
    i = pairs(k, 1); j = pairs(k, 2);
    % seq_ij: t_j >= t_i + p_i + h - M(1 - y_k) -> t_i - t_j + M y_k <= M - p_i - h
    row = row + 1;
    A(row, it(i)) = 1; A(row, it(j)) = -1; A(row, iy(k)) = M;
    b(row) = M - p(i) - h;
    % seq_ji: t_i >= t_j + p_j + h - M y_k -> -t_i + t_j - M y_k <= -(p_j + h)
    row = row + 1;
    A(row, it(i)) = -1; A(row, it(j)) = 1; A(row, iy(k)) = -M;
    b(row) = -(p(j) + h);
end

opts = optimoptions('intlinprog', 'Display', 'off');
[x, fval] = intlinprog(f, intcon, A, b, Aeq, beq, lb, ub, [], opts);

fprintf('objective = %g\n', fval);
[~, ord] = sort(x(it(1:n)));
for idx = ord'
    fprintf(' %s: enter t=%g clear C=%g\n', trains{idx}, x(it(idx)), x(iC(idx)));
end
```

## 15 · OR-Tools (C++)

model\_ortools.cpp — C++

```
// Railway single-track sequencing (big-M MILP) -- OR-Tools (C++).
// Build against OR-Tools, e.g.:
// g++ model_ortools.cpp -lortools -o railway && ./railway
#include <algorithm>
#include <iostream>
#include <memory>
#include <string>
#include <vector>

#include "ortools/linear_solver/linear_solver.h"

namespace operations_research {

void RailwayBigM() {
    const std::vector<std::string> trains = {"A", "B", "C", "D"};
    const std::vector<double> r = {0, 2, 1, 4}; // release
    const std::vector<double> p = {5, 3, 4, 2}; // running time
    const std::vector<double> w = {1, 2, 1, 3}; // weight
    const double h = 1.0; // headway
    const int n = static_cast<int>(trains.size());

    double sump = 0.0;
    for (double v : p) sump += v;
    const double M = *std::max_element(r.begin(), r.end()) + sump + (n - 1) * h; // 21

    std::unique_ptr<MPSolver> solver(MPSolver::CreateSolver("CBC"));
    const double inf = solver->infinity();

    std::vector<MPVariable*> t(n), C(n);
    for (int i = 0; i < n; ++i) {
        t[i] = solver->MakeNumVar(0.0, inf, "t_" + trains[i]);
        C[i] = solver->MakeNumVar(0.0, inf, "C_" + trains[i]);
    }
    std::vector<std::vector<MPVariable*>> y(n, std::vector<MPVariable*>(n, nullptr));
    for (int i = 0; i < n; ++i)
        for (int j = i + 1; j < n; ++j)
            y[i][j] = solver->MakeBoolVar("y_" + trains[i] + "_" + trains[j]);

    MPObjective* obj = solver->MutableObjective();
    for (int i = 0; i < n; ++i) obj->SetCoefficient(C[i], w[i]);
    obj->SetMinimization();

    for (int i = 0; i < n; ++i) {
        // release: t_i >= r_i
        MPConstraint* rel = solver->MakeRowConstraint(r[i], inf);
        rel->SetCoefficient(t[i], 1.0);
        // completion: C_i - t_i = p_i
        MPConstraint* comp = solver->MakeRowConstraint(p[i], p[i]);
        comp->SetCoefficient(C[i], 1.0);
        comp->SetCoefficient(t[i], -1.0);
    }
    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n; ++j) {
            // seq_ij: t_j - t_i - M*y_ij >= p_i + h - M
            MPConstraint* c1 = solver->MakeRowConstraint(p[i] + h - M, inf);
            c1->SetCoefficient(t[j], 1.0);
            c1->SetCoefficient(t[i], -1.0);
            c1->SetCoefficient(y[i][j], -M);
            // seq_ji: t_i - t_j + M*y_ij >= p_j + h
            MPConstraint* c2 = solver->MakeRowConstraint(p[j] + h, inf);
            c2->SetCoefficient(t[i], 1.0);
            c2->SetCoefficient(t[j], -1.0);
            c2->SetCoefficient(y[i][j], M);
        }
    }

    if (solver->Solve() == MPSolver::OPTIMAL) {
        std::cout << "objective = " << obj->Value() << "\n";
        for (int i = 0; i < n; ++i)
            std::cout << " " << trains[i] << ": enter t=" << t[i]->solution_value()
                << " clear C=" << C[i]->solution_value() << "\n";
    }
}

} // namespace operations_research

int main() {
    operations_research::RailwayBigM();
    return 0;
}
```



## 16 · OR-Tools (Java)

ModelOrTools.java — Java

```
// Railway single-track sequencing (big-M MILP) -- OR-Tools (Java).
// javac -cp ortools.jar ModelOrTools.java
// java -cp .:ortools.jar ModelOrTools
import com.google.ortools.Loader;
import com.google.ortools.linearsolver.MPConstraint;
import com.google.ortools.linearsolver.MPObjective;
import com.google.ortools.linearsolver.MPSolver;
import com.google.ortools.linearsolver.MPVariable;

public class ModelOrTools {
    public static void main(String[] args) {
        Loader.loadNativeLibraries();

        String[] trains = {"A", "B", "C", "D"};
        double[] r = {0, 2, 1, 4}; // release
        double[] p = {5, 3, 4, 2}; // running time
        double[] w = {1, 2, 1, 3}; // weight
        double h = 1.0; // headway
        int n = trains.length;

        double maxr = r[0], sump = 0;
        for (double v : r) maxr = Math.max(maxr, v);
        for (double v : p) sump += v;
        double M = maxr + sump + (n - 1) * h; // big-M = 21

        MPSolver solver = MPSolver.createSolver("CBC");
        double inf = Double.POSITIVE_INFINITY;

        MPVariable[] t = new MPVariable[n];
        MPVariable[] C = new MPVariable[n];
        for (int i = 0; i < n; i++) {
            t[i] = solver.makeNumVar(0.0, inf, "t_" + trains[i]);
            C[i] = solver.makeNumVar(0.0, inf, "C_" + trains[i]);
        }
        MPVariable[][] y = new MPVariable[n][n];
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                y[i][j] = solver.makeBoolVar("y_" + trains[i] + "_" + trains[j]);

        MPObjective obj = solver.objective();
        for (int i = 0; i < n; i++) obj.setCoefficient(C[i], w[i]);
        obj.setMinimization();

        for (int i = 0; i < n; i++) {
            MPConstraint rel = solver.makeConstraint(r[i], inf); // t_i >= r_i
            rel.setCoefficient(t[i], 1.0);
            MPConstraint comp = solver.makeConstraint(p[i], p[i]); // C_i - t_i = p_i
            comp.setCoefficient(C[i], 1.0);
            comp.setCoefficient(t[i], -1.0);
        }
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                // seq_ij: t_j - t_i - M*y_ij >= p_i + h - M
                MPConstraint c1 = solver.makeConstraint(p[i] + h - M, inf);
                c1.setCoefficient(t[j], 1.0);
                c1.setCoefficient(t[i], -1.0);
                c1.setCoefficient(y[i][j], -M);
                // seq_ji: t_i - t_j + M*y_ij >= p_j + h
                MPConstraint c2 = solver.makeConstraint(p[j] + h, inf);
                c2.setCoefficient(t[i], 1.0);
                c2.setCoefficient(t[j], -1.0);
                c2.setCoefficient(y[i][j], M);
            }
        }

        if (solver.solve() == MPSolver.ResultStatus.OPTIMAL) {
            System.out.println("objective = " + obj.value());
            for (int i = 0; i < n; i++) {
                System.out.printf(" %s: enter t=%.0f clear C=%.0f\n",
                    trains[i], t[i].solutionValue(), C[i].solutionValue());
            }
        }
    }
}
```

## 17 · AMPL

model\_ampl.mod — algebraic modelling language

---

```
# Railway single-track sequencing (big-M MILP) -- AMPL.
# Self-contained (model + data + solve). Run with:
#     ampl model_ampl.mod

set TRAINS ordered;
param r{TRAINS} >= 0;      # release (earliest entry)
param p{TRAINS} > 0;       # running time on the segment
param w{TRAINS} >= 0;      # priority weight
param h >= 0;              # minimum headway
param M := (max{i in TRAINS} r[i]) + (sum{i in TRAINS} p[i]) + (card(TRAINS) - 1) * h; # big-M

var t{TRAINS} >= 0;        # entry time
var C{TRAINS} >= 0;        # clearance time
var y{i in TRAINS, j in TRAINS: ord(i) < ord(j)} binary; # y[i,j]=1 iff i before j

minimize weighted_clearance: sum{i in TRAINS} w[i] * C[i];

subject to release {i in TRAINS}: t[i] >= r[i];
subject to completion {i in TRAINS}: C[i] = t[i] + p[i];
subject to seq_ij {i in TRAINS, j in TRAINS: ord(i) < ord(j)}:
    t[j] >= t[i] + p[i] + h - M * (1 - y[i,j]);
subject to seq_ji {i in TRAINS, j in TRAINS: ord(i) < ord(j)}:
    t[i] >= t[j] + p[j] + h - M * y[i,j];

data;
set TRAINS := A B C D;
param:   r   p   w :=
    A   0   5   1
    B   2   3   2
    C   1   4   1
    D   4   2   3 ;
param h := 1;

option solver cbc;          # or gurobi, cplex, highs, ...
solve;
display weighted_clearance;
display t, C, y;
```

## 18 · GAMS

model\_gams.gms — algebraic modelling language

---

```
$title Railway single-track sequencing (big-M MILP) -- GAMS
* Run with:  gams model_gams.gms

Set i 'trains' / A, B, C, D /;
Alias (i, j);

Parameters
  r(i) 'release (earliest entry)' / A 0, B 2, C 1, D 4 /
  p(i) 'running time on the segment' / A 5, B 3, C 4, D 2 /
  w(i) 'priority weight' / A 1, B 2, C 1, D 3 /;

Scalar h 'minimum headway' / 1 /;
Scalar M 'big-M constant';
M = smax(i, r(i)) + sum(i, p(i)) + (card(i) - 1) * h;  display M;

Variables          z 'weighted clearance';
Positive Variables t(i) 'entry time', C(i) 'clearance time';
Binary Variables   y(i,j) 'y(i,j)=1 iff i enters before j';

Equations
  obj              'objective'
  release(i)       'earliest entry'
  completion(i)    'clearance = entry + running time'
  seq_ij(i,j)      'i before j'
  seq_ji(i,j)      'j before i';

obj..              z =e= sum(i, w(i) * C(i));
release(i)..       t(i) =g= r(i);
completion(i)..    C(i) =e= t(i) + p(i);
seq_ij(i,j)$ (ord(i) < ord(j)).. t(j) =g= t(i) + p(i) + h - M * (1 - y(i,j));
seq_ji(i,j)$ (ord(i) < ord(j)).. t(i) =g= t(j) + p(j) + h - M * y(i,j);

Model railway / all /;
solve railway using mip minimizing z;

display z.l, t.l, C.l, y.l;
```

## 19 · GNU MathProg (GMPL)

model\_glpk.mod — glpsol

```
/* Railway single-track sequencing (big-M MILP) -- GNU MathProg (GMPL).
   Fully self-contained. Solve with:
   glpsol --model model_glpk.mod
*/

param n := 4;
set I := 1..n;                                /* trains, by position */
param nm{I}, symbolic;                        /* train names */
param r{I};                                   /* release (earliest entry) */
param p{I};                                   /* running time on the segment */
param w{I};                                   /* priority weight */
param h := 1;                                 /* minimum headway */
param M := (max{i in I} r[i]) + (sum{i in I} p[i]) + (n - 1) * h; /* big-M = 21 */

var t{I} >= 0;                                /* entry time */
var C{I} >= 0;                                /* clearance time */
var y{i in I, j in I: i < j}, binary;        /* y[i,j]=1 iff i enters before j */

minimize weighted_clearance: sum{i in I} w[i] * C[i];

s.t. release{i in I}: t[i] >= r[i];
s.t. completion{i in I}: C[i] = t[i] + p[i];
s.t. seq_ij{i in I, j in I: i < j}: t[j] >= t[i] + p[i] + h - M * (1 - y[i,j]);
s.t. seq_ji{i in I, j in I: i < j}: t[i] >= t[j] + p[j] + h - M * y[i,j];

solve;

printf "objective = %g\n", sum{i in I} w[i] * C[i];
printf {i in I} " %s: enter t=%g clear C=%g\n", nm[i], t[i], C[i];

data;
param nm := 1 A 2 B 3 C 4 D;
param r := 1 0 2 2 3 1 4 4;
param p := 1 5 2 3 3 4 4 2;
param w := 1 1 2 2 3 1 4 3;

end;
```

## 20 · ZIMPL

model\_zimpl.zpl — algebraic modelling language

---

```
# Railway single-track sequencing (big-M MILP) -- ZIMPL.
# Generate an LP/MPS and solve, e.g.:
#   zimpl model_zimpl.zpl          # writes model_zimpl.lp
#   glpsol --lp model_zimpl.lp

set I := { 1 .. 4 };                # trains, by position
param nm[I] := <1> "A", <2> "B", <3> "C", <4> "D";
param r[I] := <1> 0, <2> 2, <3> 1, <4> 4;    # release
param p[I] := <1> 5, <2> 3, <3> 4, <4> 2;    # running time
param w[I] := <1> 1, <2> 2, <3> 1, <4> 3;    # weight
param h := 1;                       # headway
param M := (max <i> in I : r[i]) + (sum <i> in I : p[i]) + (card(I) - 1) * h; # big-M

var t[I] >= 0;                      # entry time
var C[I] >= 0;                      # clearance time
var y[<i,j> in I * I with i < j] binary; # y[i,j]=1 iff i enters before j

minimize weighted_clearance: sum <i> in I : w[i] * C[i];

subto release: forall <i> in I : t[i] >= r[i];
subto completion: forall <i> in I : C[i] == t[i] + p[i];
subto seq_ij: forall <i,j> in I * I with i < j :
    t[j] >= t[i] + p[i] + h - M * (1 - y[i,j]);
subto seq_ji: forall <i,j> in I * I with i < j :
    t[i] >= t[j] + p[j] + h - M * y[i,j];
```

## 21 · CPLEX OPL

model\_opl.mod — IBM modelling language (oplrun)

---

```
// Railway single-track sequencing (big-M MILP) -- CPLEX OPL.
// Run in IBM ILOG CPLEX Optimization Studio: oplrun model_opl.mod
// (Self-contained: data is inline, so no .dat file is needed.)

{string} TRAINS = {"A", "B", "C", "D"};           // trains, in order
int r[TRAINS] = ["A":0, "B":2, "C":1, "D":4];     // release
int p[TRAINS] = ["A":5, "B":3, "C":4, "D":2];     // running time
int w[TRAINS] = ["A":1, "B":2, "C":1, "D":3];     // weight
int h = 1;                                         // headway
int M = (max(i in TRAINS) r[i]) + (sum(i in TRAINS) p[i]) + (card(TRAINS) - 1) * h; // big-M = 21

dvar float+ t[TRAINS];                           // entry time
dvar float+ C[TRAINS];                           // clearance time
dvar boolean y[TRAINS][TRAINS]; // y[i][j]=1 iff i enters before j (i before j only)

minimize sum(i in TRAINS) w[i] * C[i];

subject to {
  forall(i in TRAINS) release: t[i] >= r[i];
  forall(i in TRAINS) completion: C[i] == t[i] + p[i];
  forall(ordered i, j in TRAINS) {
    t[j] >= t[i] + p[i] + h - M * (1 - y[i][j]); // i before j
    t[i] >= t[j] + p[j] + h - M * y[i][j];      // j before i
  }
}

execute DISPLAY {
  writeln("objective = ", cplex.getObjValue());
  for (var i in TRAINS)
    writeln(" ", i, ": enter t=", t[i], " clear C=", C[i]);
}
```

## 22 · MiniZinc

model\_minizinc.mzn — constraint/MIP modelling

---

```
% Railway single-track sequencing (big-M MILP) -- MiniZinc.
% Solve with a MIP backend so the big-M form is kept linear, e.g.:
%   minimzinc --solver coin-bc model_minizinc.mzn

enum TRAINS = { A, B, C, D };
array[TRAINS] of int: r = [0, 2, 1, 4]; % release (earliest entry)
array[TRAINS] of int: p = [5, 3, 4, 2]; % running time on the segment
array[TRAINS] of int: w = [1, 2, 1, 3]; % priority weight
int: h = 1; % minimum headway
int: n = card(TRAINS);
int: M = max(r) + sum(p) + (n - 1) * h; % big-M = 21
int: ub = M + max(p); % safe horizon for the clocks

array[TRAINS] of var 0..ub: t; % entry time
array[TRAINS] of var 0..ub: C; % clearance time
array[TRAINS, TRAINS] of var 0..1: y; % y[i,j]=1 iff i before j (used for i<j)

constraint forall(i in TRAINS)(t[i] >= r[i]);
constraint forall(i in TRAINS)(C[i] = t[i] + p[i]);
constraint forall(i, j in TRAINS where i < j)(
    t[j] >= t[i] + p[i] + h - M * (1 - y[i,j]) /\
    t[i] >= t[j] + p[j] + h - M * y[i,j]
);

var int: cost = sum(i in TRAINS)(w[i] * C[i]);
solve minimize cost;

output ["cost = \"(cost)\""] ++
    [" \"(i): enter t=\"(t[i])\" clear C=\"(C[i])\" | i in TRAINS];
```

## 23 · CPLEX LP format

model.lp — portable exchange format

---

```
\* railway_bigm *\
Minimize
OBJ: C_A + 2 C_B + C_C + 3 C_D
Subject To
completion_A: C_A - t_A = 5
completion_B: C_B - t_B = 3
completion_C: C_C - t_C = 4
completion_D: C_D - t_D = 2
release_A: t_A >= 0
release_B: t_B >= 2
release_C: t_C >= 1
release_D: t_D >= 4
seq_A_before_B: - t_A + t_B - 21 y_A_B >= -15
seq_A_before_C: - t_A + t_C - 21 y_A_C >= -15
seq_A_before_D: - t_A + t_D - 21 y_A_D >= -15
seq_B_before_A: t_A - t_B + 21 y_A_B >= 4
seq_B_before_C: - t_B + t_C - 21 y_B_C >= -17
seq_B_before_D: - t_B + t_D - 21 y_B_D >= -17
seq_C_before_A: t_A - t_C + 21 y_A_C >= 5
seq_C_before_B: t_B - t_C + 21 y_B_C >= 5
seq_C_before_D: - t_C + t_D - 21 y_C_D >= -16
seq_D_before_A: t_A - t_D + 21 y_A_D >= 3
seq_D_before_B: t_B - t_D + 21 y_B_D >= 3
seq_D_before_C: t_C - t_D + 21 y_C_D >= 3
Binaries
y_A_B
y_A_C
y_A_D
y_B_C
y_B_D
y_C_D
End
```



## 24 · MPS format

model.mps — portable exchange format

```
*SENSE:Minimize
NAME          railway_bigm
ROWS
N  OBJ
G  release_A
E  completion_A
G  release_B
E  completion_B
G  release_C
E  completion_C
G  release_D
E  completion_D
G  seq_A_before_B
G  seq_B_before_A
G  seq_A_before_C
G  seq_C_before_A
G  seq_A_before_D
G  seq_D_before_A
G  seq_B_before_C
G  seq_C_before_B
G  seq_B_before_D
G  seq_D_before_B
G  seq_C_before_D
G  seq_D_before_C
COLUMNS
C_A      completion_A  1.000000000000e+00
C_A      OBJ           1.000000000000e+00
C_B      completion_B  1.000000000000e+00
C_B      OBJ           2.000000000000e+00
C_C      completion_C  1.000000000000e+00
C_C      OBJ           1.000000000000e+00
C_D      completion_D  1.000000000000e+00
C_D      OBJ           3.000000000000e+00
T_A      release_A     1.000000000000e+00
T_A      completion_A -1.000000000000e+00
T_A      seq_A_before_B -1.000000000000e+00
T_A      seq_B_before_A -1.000000000000e+00
T_A      seq_A_before_C -1.000000000000e+00
T_A      seq_C_before_A -1.000000000000e+00
T_A      seq_A_before_D -1.000000000000e+00
T_A      seq_D_before_A -1.000000000000e+00
T_B      release_B     1.000000000000e+00
T_B      completion_B -1.000000000000e+00
T_B      seq_A_before_B 1.000000000000e+00
T_B      seq_B_before_A -1.000000000000e+00
T_B      seq_B_before_C -1.000000000000e+00
T_B      seq_C_before_B 1.000000000000e+00
T_B      seq_B_before_D -1.000000000000e+00
T_B      seq_D_before_B 1.000000000000e+00
T_C      release_C     1.000000000000e+00
T_C      completion_C -1.000000000000e+00
T_C      seq_A_before_C 1.000000000000e+00
T_C      seq_C_before_A -1.000000000000e+00
T_C      seq_B_before_C 1.000000000000e+00
T_C      seq_C_before_B -1.000000000000e+00
T_C      seq_C_before_D -1.000000000000e+00
T_C      seq_D_before_C 1.000000000000e+00
T_D      release_D     1.000000000000e+00
T_D      completion_D -1.000000000000e+00
T_D      seq_A_before_D 1.000000000000e+00
T_D      seq_D_before_A -1.000000000000e+00
T_D      seq_B_before_D 1.000000000000e+00
T_D      seq_D_before_B -1.000000000000e+00
T_D      seq_C_before_D 1.000000000000e+00
T_D      seq_D_before_C -1.000000000000e+00
MARK      'MARKER'      'INTORG'
Y_A_B     seq_A_before_B -2.100000000000e+01
Y_A_B     seq_B_before_A 2.100000000000e+01
MARK      'MARKER'      'INTEND'
MARK      'MARKER'      'INTORG'
Y_A_C     seq_A_before_C -2.100000000000e+01
Y_A_C     seq_C_before_A 2.100000000000e+01
MARK      'MARKER'      'INTEND'
MARK      'MARKER'      'INTORG'
Y_A_D     seq_A_before_D -2.100000000000e+01
Y_A_D     seq_D_before_A 2.100000000000e+01
MARK      'MARKER'      'INTEND'
MARK      'MARKER'      'INTORG'
Y_B_C     seq_B_before_C -2.100000000000e+01
Y_B_C     seq_C_before_B 2.100000000000e+01
MARK      'MARKER'      'INTEND'
MARK      'MARKER'      'INTORG'
Y_B_D     seq_B_before_D -2.100000000000e+01
Y_B_D     seq_D_before_B 2.100000000000e+01
MARK      'MARKER'      'INTEND'
MARK      'MARKER'      'INTORG'
Y_C_D     seq_C_before_D -2.100000000000e+01
Y_C_D     seq_D_before_C 2.100000000000e+01
MARK      'MARKER'      'INTEND'
RHS
RHS      release_A     0.000000000000e+00
RHS      completion_A  5.000000000000e+00
RHS      release_B     2.000000000000e+00
RHS      completion_B  3.000000000000e+00
RHS      release_C     1.000000000000e+00
RHS      completion_C  4.000000000000e+00
RHS      release_D     4.000000000000e+00
RHS      completion_D  2.000000000000e+00
RHS      seq_A_before_B -1.500000000000e+01
RHS      seq_B_before_A 4.000000000000e+00
RHS      seq_A_before_C -1.500000000000e+01
RHS      seq_C_before_A 5.000000000000e+00
RHS      seq_A_before_D -1.500000000000e+01
RHS      seq_D_before_A 3.000000000000e+00
RHS      seq_B_before_C -1.700000000000e+01
RHS      seq_C_before_B 5.000000000000e+00
RHS      seq_B_before_D -1.700000000000e+01
RHS      seq_D_before_B 3.000000000000e+00
RHS      seq_C_before_D -1.600000000000e+01
RHS      seq_D_before_C 3.000000000000e+00
BOUNDS
BV BND    y_A_B
BV BND    y_A_C
BV BND    y_A_D
BV BND    y_B_C
BV BND    y_B_D
BV BND    y_C_D
ENDATA
```