Basic information

In [4]:
```python
import gurobipy as gp
from gurobipy import GRB
import numpy as np
import random
import pandas as pd

# random seed for reproducibility
random.seed(42)
np.random.seed(42)

# load in population per LGA for 2001
data = pd.read_excel('2001 population.xlsx')

# define the LGAs
lgas_nsw = data['LGA name'].tolist()

# define the demographic groups for all LGAs
demographic_groups = [
    "0-4yrs old", "5-9yrs old", "10-14yrs old", "15-19yrs old", "20-24yrs old",
    "25-29yrs old", "30-34yrs old", "35-39yrs old", "40-44yrs old", "45-49yrs old",
    "50-54yrs old", "55-59yrs old", "60-64yrs old", "65-69yrs old", "70-74yrs old",
    "75-79yrs old", "80-84yrs old", "85 and over"
]

# generate demographic groups for each LGA
population_data = {lga: {} for lga in lgas_nsw}
# generate random age breakdowns based on population data for each LGA
for lga in lgas_nsw:
    total_population = data[data['LGA name'] == lga]['Total persons no.'].values[0]
    remaining_population = total_population

    population_data[lga] = {}
    for group in demographic_groups:
        if group == demographic_groups[-1]:
            population_data[lga][group] = remaining_population
        else:
            min_population = 0
            max_population = min(remaining_population, total_population // 5)  # ensure the group doesn't exceed 20% of the total population
            population_data[lga][group] = random.randint(min_population, max_population)
            remaining_population -= population_data[lga][group]

# (RDI) data for different age groups
# the RDI data is approximate from the RDI data given in the report converted into the same demographic groups with the census data
rdi_data = {
    "0-4yrs old": 500,
    "5-9yrs old": 700,
    "10-14yrs old": 1000,
    "15-19yrs old": 1300,
    "20-24yrs old": 1000,
    "25-29yrs old": 1000,
    "30-34yrs old": 1000,
    "35-39yrs old": 1000,
    "40-44yrs old": 1000,
    "45-49yrs old": 1000,
    "50-54yrs old": 1300,
    "55-59yrs old": 1300,
    "60-64yrs old": 1300,
    "65-69yrs old": 1300,
    "70-74yrs old": 1300,
    "75-79yrs old": 1300,
    "80-84yrs old": 1300,
    "85 and over": 1300
}

# total daily intake of calcium(mg) demand for each LGA
total_milk_demand_by_lga = {lga: 0 for lga in lgas_nsw}

for lga in lgas_nsw:
    total_demand = sum(population_data[lga][age_group] * rdi_data[age_group] for age_group in demographic_groups)
    total_milk_demand_by_lga[lga] = total_demand

# convert calcium(mg) demand into to liters (milk demand)
for lga, total_demand_mg in total_milk_demand_by_lga.items():
    total_demand_liters = total_demand_mg * 1.0E-6
    total_milk_demand_by_lga[lga] = total_demand_liters

# generate random distance data
distance_data = {(i, j): np.random.randint(1, 200) for i in lgas_nsw for j in lgas_nsw if i != j}
```

Part 1a

In [2]:
```python
model = gp.Model("MilkDistributionWithTankers")
model.setParam('TimeLimit', 600)

# decision variables
x = model.addVars(lgas_nsw, lgas_nsw, vtype=GRB.BINARY, name="x")
y = model.addVars(lgas_nsw, vtype=GRB.BINARY, name="y")
n = model.addVars(lgas_nsw, vtype=GRB.INTEGER, name="n")
t = model.addVars(lgas_nsw, vtype=GRB.INTEGER, name="t")

# objective func
obj = (
    gp.quicksum(distance_data[i, j] * x[i, j] * total_milk_demand_by_lga[j] * 0.10 for i in lgas_nsw for j in lgas_nsw if i != j) +
    gp.quicksum(n[i] * 1000 for i in lgas_nsw) +  # cost of creating distribution centers
    gp.quicksum(t[i] * 125000 for i in lgas_nsw)  # cost of purchasing tankers
)
model.setObjective(obj, GRB.MINIMIZE)

# constraints
# each LGA should receive its total milk demand
for j in lgas_nsw:
    model.addConstr(gp.quicksum(x[i, j] for i in lgas_nsw) == 1, name=f"Meet_Demand_{j}")

# each distribution center can supply milk to LGAs if it's built
for i in lgas_nsw:
    for j in lgas_nsw:
        model.addConstr(x[i, j] <= y[i], name=f"Center_Supply_{i}_{j}")

# total distribution capacity of a center in LGA i must not exceed 50% of the state's demand
for i in lgas_nsw:
    model.addConstr(n[i] <= 0.5 * gp.quicksum(total_milk_demand_by_lga[j] for j in lgas_nsw), name=f"Capacity_{i}")

# max of 3 distribution centers can be built
model.addConstr(gp.quicksum(y[i] for i in lgas_nsw) <= 3, name="Max_Centers")

# each distribution center can service up to 4 LGAs with tankers
for i in lgas_nsw:
    model.addConstr(
        gp.quicksum(x[i, j] for j in lgas_nsw) <= 4 * t[i], name=f"Tanker_Service_Limit_{i}")

# tanker capacity should not be exceeded
for i in lgas_nsw:
    model.addConstr(
        gp.quicksum(x[i, j] for j in lgas_nsw) <= t[i] * 10000, name=f"Tanker_Capacity_Limit_{i}")

model.update()
model.optimize()
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2024-08-02
Set parameter TimeLimit to value 600
Gurobi Optimizer version 10.0.2 build v10.0.2rc0 (mac64[rosetta2])

CPU model: Apple M1
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 17158 rows, 17028 columns and 83721 nonzeros
Model fingerprint: 0xea202f51
Variable types: 0 continuous, 17028 integer (16770 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+04]
  Objective range  [2e-01, 1e+05]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 3e+03]
Presolve removed 258 rows and 129 columns
Presolve time: 0.13s
Presolved: 16900 rows, 16899 columns, 66822 nonzeros
Variable types: 0 continuous, 16899 integer (16770 binary)
Found heuristic solution: objective 4156101.7614

Root relaxation: objective 4.046338e+06, 14545 iterations, 0.95 seconds (2.15 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 4046338.33    0 1179 4156101.76 4046338.33  2.64%     -    1s
H    0     0                    4145974.3089 4046338.33  2.40%     -    1s
     0     0 4046353.92    0 1291 4145974.31 4046353.92  2.40%     -    1s
     0     0 4046372.41    0 1429 4145974.31 4046372.41  2.40%     -    3s
H    0     0                    4145626.4042 4046372.41  2.39%     -    6s
     0     0 4046373.07    0 1434 4145626.40 4046373.07  2.39%     -    6s
     0     0 4046373.12    0 1430 4145626.40 4046373.12  2.39%     -    6s
     0     0 4046405.72    0 1439 4145626.40 4046405.72  2.39%     -    7s
     0     0 4046405.74    0 1437 4145626.40 4046405.74  2.39%     -    7s
     0     0 4046423.40    0 1561 4145626.40 4046423.40  2.39%     -    9s
     0     0 4046424.07    0 1554 4145626.40 4046424.07  2.39%     -    9s
     0     0 4046433.84    0 1610 4145626.40 4046433.84  2.39%     -    9s
     0     0 4046434.53    0 1619 4145626.40 4046434.53  2.39%     -    9s
     0     0 4046438.37    0 1694 4145626.40 4046438.37  2.39%     -   10s
     0     0 4046438.37    0 1694 4145626.40 4046438.37  2.39%     -   10s
     0     2 4046438.37    0 1694 4145626.40 4046438.37  2.39%     -   11s
    31    40 4046458.57    5 1467 4145626.40 4046448.71  2.39%   576   19s
H   39    48                    4145531.1410 4046448.71  2.39%   566   20s
    71   114 4046468.08    7 1431 4145531.14 4046448.71  2.39%   553   33s
H   80   114                    4145348.7149 4046448.71  2.39%   537   33s
   113   306 4046491.92   12 1284 4145348.71 4046448.71  2.39%   529   41s
   305   613 4046539.23   26 1301 4145348.71 4046448.71  2.39%   470   56s
   612  1140 4046581.02   42 1354 4145348.71 4046448.71  2.39%   443   79s
  1177  1297 4046927.27   77 1253 4145348.71 4046448.71  2.39%   478  102s
  1358  2081     cutoff   90      4145348.71 4046448.84  2.39%   565  128s
  2594  2410 4058207.32   32  370 4145348.71 4046450.27  2.39%   481  148s
H 2689  2410                    4145235.8329 4046452.84  2.38%   491  148s
  3163  2411 4053587.02   12 1694 4145235.83 4046455.34  2.38%   501  154s
  3165  2412 4062646.02   45 1179 4145235.83 4046455.34  2.38%   501  156s
  3166  2413 4050046.01   42 1522 4145235.83 4046455.34  2.38%   500  160s
  3169  2415 4079791.18   93 1790 4145235.83 4046455.34  2.38%   500  167s
  3172  2417 4067847.28  120 1746 4145235.83 4046458.86  2.38%   499  171s
  3174  2418 4056192.39   25 1743 4145235.83 4046459.49  2.38%   499  178s
  3175  2419 4046556.58   34 1743 4145235.83 4046459.49  2.38%   499  180s
  3176  2420 4056273.45  165 1743 4145235.83 4046459.49  2.38%   499  200s
  3177  2423 4046462.33   10 1781 4145235.83 4046459.49  2.38%   505  208s
  3183  2433 4046466.48   12 1699 4145235.83 4046461.21  2.38%   505  210s
  3232  2466 4048253.75   15  905 4145235.83 4046465.91  2.38%   508  215s
  3249  2493 4046511.77   16 1569 4145235.83 4046465.91  2.38%   513  224s
  3275  2557 4046595.94   18 1464 4145235.83 4046465.91  2.38%   517  229s
  3359  2624 4046710.95   19 1470 4145235.83 4046465.91  2.38%   522  233s
  3482  2648 4046770.65   21 1425 4145235.83 4046465.91  2.38%   525  238s
  3587  2622 4046781.20   23 1392 4145235.83 4046465.91  2.38%   534  248s
  3596  2688 4049711.74   23  855 4145235.83 4046465.91  2.38%   542  253s
  3679  2826 4046950.95   26 1438 4145235.83 4046465.91  2.38%   560  258s
  3847  2895 4047078.18   29 1471 4145235.83 4046465.91  2.38%   562  263s
  3990  2923 4047084.16   31 1547 4145235.83 4046466.60  2.38%   569  271s
  4092  2957 4056762.05   15    1 4145235.83 4046466.60  2.38%   580  277s
  4240  3080 4057091.31   24  200 4145235.83 4046466.60  2.38%   591  284s
  4476  3111 4057621.73   29    1 4145235.83 4046466.60  2.38%   593  288s
  4666  3207 4058290.47   33    1 4145235.83 4046466.60  2.38%   599  310s
  4903  3216 4058779.42   36  230 4145235.83 4046466.60  2.38%   595  320s
  5090  3309 4060164.12   54  227 4145235.83 4046466.60  2.38%   600  330s
  5361  3311 4060869.72   81  214 4145235.83 4046466.60  2.38%   597  338s
H 5505  3141                    4144458.4748 4046466.60  2.36%   608  338s
  5518  3249 infeasible   95      4144458.47 4046466.60  2.36%   609  349s
  5730  3202 4064571.50  100  221 4144458.47 4046466.60  2.36%   619  356s
  5841  3243 4067105.12  107  231 4144458.47 4046466.60  2.36%   633  363s
H 5873  3118                    4144357.5894 4046466.60  2.36%   634  363s
  5973  3181 4068416.61  109  222 4144357.59 4046470.85  2.36%   643  370s
H 6007  3061                    4144352.9259 4046470.88  2.36%   644  370s
  6128  3107 4046797.49   14 1183 4144352.93 4046514.12  2.36%   653  378s
  6276  3211 4051242.67   15  259 4144352.93 4046528.11  2.36%   664  386s
  6540  3221     cutoff   38      4144352.93 4046543.58  2.36%   664  396s
  6775  3266 4050238.63   19 1144 4144352.93 4046553.96  2.36%   669  403s
  7045  3328 4084830.06   34    1 4144352.93 4046626.69  2.36%   676  411s
  7336  3383 4072469.68   25    1 4144352.93 4046659.46  2.36%   679  418s
  7605  3531 4054712.92   25  255 4144352.93 4046799.21  2.35%   684  428s
  7966  3682     cutoff   26      4144352.93 4046839.23  2.35%   681  438s
  8343  3805 4054725.95   21    1 4144352.93 4046848.25  2.35%   674  449s
  8710  3721 4055453.29   31    1 4144352.93 4046852.82  2.35%   667  459s
  8920  3746 4066483.48   26    1 4144352.93 4046944.01  2.35%   674  470s
  9146  3700     cutoff   38      4144352.93 4047019.68  2.35%   685  480s
  9329  3691 4059361.72   38    1 4144352.93 4047088.47  2.35%   700  493s
  9455  3697 4061359.19   37  469 4144352.93 4047107.96  2.35%   709  506s
H 9536  3697                    4143722.3086 4047107.96  2.33%   709  506s
  9674  3769 4053565.04   32  350 4143722.31 4047112.03  2.33%   720  517s
  9931  3760 4051239.94   41  935 4143722.31 4047189.01  2.33%   732  529s
 10042  3784 4051680.71   20  248 4143722.31 4047283.62  2.33%   742  540s
 10216  3850     cutoff   45      4143722.31 4047409.50  2.32%   757  552s
 10442  3875 4053587.23   51  460 4143722.31 4047558.42  2.32%   769  600s

Cutting planes:
  Gomory: 2
  MIR: 4
  Flow cover: 2
  Zero half: 52

Explored 10645 nodes (8314103 simplex iterations) in 600.36 seconds (1057.50 work units)
Thread count was 8 (of 8 available processors)

Solution count 10: 4.14372e+06 4.14435e+06 4.14436e+06 ... 4.1561e+06

Time limit reached
Best objective 4.143722308580e+06, best bound 4.047626885810e+06, gap 2.3191%
```

In [3]:
```python
# results
print("Selected Distribution Centers:")
for i in lgas_nsw:
    if y[i].x > 0.5:
        print(f"Build a center in {i}")
print('----------------------------------------')

print("Number of Tankers for Each Distribution Center:")
for i in lgas_nsw:
    num_tankers = t[i].x
    if num_tankers > 0:
        print(f"Distribution Center in {i} should purchase {num_tankers} tankers")

Minimum_total_cost = model.objVal
print('----------------------------------------')
print(f"Minimum total cost: ${Minimum_total_cost:.2f}")
```

```
Selected Distribution Centers:
Build a center in Sydney
Build a center in Temora
Build a center in Woollahra
----------------------------------------
Number of Tankers for Each Distribution Center:
Distribution Center in Sydney should purchase 11.0 tankers
Distribution Center in Temora should purchase 12.0 tankers
Distribution Center in Woollahra should purchase 10.0 tankers
----------------------------------------
Minimum total cost: $4143722.31
```

brief explanation

In this model, I have addressed a milk distribution problem where the goal is to minimize the overall cost while determining out where distribution centers are located and how many tankers each distribution center has to buy within a set of restrictions. I've also set a limitation to the run time for 10 minutes since the model is running for too long. Here's a quick rundown of what has been done:

Decision Variables:
Binary variables x[i, j] indicate the assignment of LGAs (j) to distribution centers (i).
Binary variables y[i] represent the number of distribution center if it is built in LGA i.
Integer variables n[i] represent the capacity of distribution center that are built in LGA i.
Integer variables t[i] represent the number of tankers at each distribution center.

Objective function:
The main goal is to minimize the total costs, which include the cost of creating distribution centers, the cost of distribution, and the cost of purchasing new tankers for each distribution center. The specific components of the objective function are:
The cost of distribution, which depends on the distance traveled, the milk demand, and a unit cost of 0.10 dollar.
The cost of creating distribution centers, with a fixed cost of 1,000 dollar per center.
The cost of purchasing tankers, with a fixed cost of 125,000 dollar per tanker.

Constraints:
"Meet_Demand": Ensures that each LGA receives its total milk demand by setting constraints on the assignment of LGAs to distribution centers.
"Center_Supply": Ensures that distribution centers can supply milk to LGAs only if they are built.
"Capacity": Sets a constraint to limit the total distribution capacity of a center in LGA i to not exceed 50 percent of the state's demand.
"Max_Centers": Restricts the number of distribution centers to be at most three.
"Tanker_Service_Limit": Ensures that each distribution center can serve up to four LGAs with one tankers.
"Tanker_Capacity_Limit": Restricts the capacity of each tanker so that it is not exceeded.

Results: Three distribution centres should be built in Sydney, Temora, and Woollahra, according to the output. It is advised that these centres buy 10 tankers for Woollahra, 12 tankers for Temora, and 11 tankers for Sydney. The optimal solution comes with a 4,143,722.31 dollar minimal total cost.

Part 2a

In [11]:
```python
model = gp.Model("MilkDistributionWithTankersTimeMinimization")

# decision variables
x = model.addVars(lgas_nsw, lgas_nsw, vtype=GRB.BINARY, name="x")
y = model.addVars(lgas_nsw, vtype=GRB.BINARY, name="y")
n = model.addVars(lgas_nsw, vtype=GRB.INTEGER, name="n")
t = model.addVars(lgas_nsw, vtype=GRB.INTEGER, name="t")

# new objective func to minimize total distribution route time
obj = gp.quicksum((distance_data[i, j] / 40 + 1.2 * n[i]) * x[i, j] * total_milk_demand_by_lga[j] for i in lgas_nsw for j in lgas_nsw if i != j)
model.setObjective(obj, GRB.MINIMIZE)

# constraints
# each LGA should receive its total milk demand
for j in lgas_nsw:
    model.addConstr(gp.quicksum(x[i, j] for i in lgas_nsw) == 1, name=f"Meet_Demand_{j}")

# each distribution center can supply milk to LGAs if it's built
for i in lgas_nsw:
    for j in lgas_nsw:
        model.addConstr(x[i, j] <= y[i], name=f"Center_Supply_{i}_{j}")

# total distribution capacity of a center in LGA i must not exceed 50% of the state's demand
for i in lgas_nsw:
    model.addConstr(n[i] <= 0.5 * gp.quicksum(total_milk_demand_by_lga[j] for j in lgas_nsw), name=f"Capacity_{i}")

# max of 3 distribution centers can be built
model.addConstr(gp.quicksum(y[i] for i in lgas_nsw) <= 3, name="Max_Centers")

# each distribution center can service up to 4 LGAs with tankers
for i in lgas_nsw:
    model.addConstr(gp.quicksum(x[i, j] for j in lgas_nsw) <= 4 * t[i], name=f"Tanker_Service_Limit_{i}")

# tanker capacity should not be exceeded
for i in lgas_nsw:
    model.addConstr(gp.quicksum(x[i, j] for j in lgas_nsw) <= t[i] * 10000, name=f"Tanker_Capacity_Limit_{i}")

model.update()
model.optimize()
```

```
Gurobi Optimizer version 10.0.2 build v10.0.2rc0 (mac64[rosetta2])

CPU model: Apple M1
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 17158 rows, 17028 columns and 83721 nonzeros
Model fingerprint: 0xfdbeecf4
Model has 16512 quadratic objective terms
Variable types: 0 continuous, 17028 integer (16770 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+04]
  Objective range  [6e-02, 2e+03]
  QObjective range [3e+00, 8e+02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 3e+03]
Presolve removed 387 rows and 258 columns
Presolve time: 0.14s
Presolved: 16771 rows, 16770 columns, 50052 nonzeros
Variable types: 0 continuous, 16770 integer (16770 binary)
Found heuristic solution: objective 7775.4403550

Root relaxation: objective 3.772082e+03, 8763 iterations, 0.44 seconds (0.98 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 3772.08159    0 1155 7775.44035 3772.08159  51.5%     -    0s
H    0     0                    5243.5772350 3772.08159  28.1%     -    0s
     0     0 3780.65433    0 1350 5243.57723 3780.65433  27.9%     -    1s
     0     0 3789.50484    0 1387 5243.57723 3789.50484  27.7%     -    2s
     0     0 3789.62083    0 1432 5243.57723 3789.62083  27.7%     -    2s
     0     0 3789.62969    0 1430 5243.57723 3789.62969  27.7%     -    2s
     0     0 3789.65972    0 1430 5243.57723 3789.65972  27.7%     -    2s
H    0     0                    4744.6997425 3947.40111  16.8%     -    3s
     0     0 3948.20831    0 1612 4744.69974 3948.20831  16.8%     -    3s
     0     0 3948.20831    0 1557 4744.69974 3948.20831  16.8%     -    3s
     0     0 4018.24009    0 1720 4744.69974 4018.24009  15.3%     -    4s
     0     0 4018.24009    0 1806 4744.69974 4018.24009  15.3%     -    4s
     0     0 4018.24009    0 1735 4744.69974 4018.24009  15.3%     -    4s
     0     0 4018.24009    0 1734 4744.69974 4018.24009  15.3%     -    4s
     0     2 4018.24009    0 1734 4744.69974 4018.24009  15.3%     -    5s
*   11     8               3    4738.8607625 4018.24009  15.2%  1775    6s
*   18    10               4    4680.3235850 4018.24009  14.1%  1473    6s
*   43    12               7    4524.6787175 4018.24009  11.2%   934    7s
   135     4     cutoff   44      4524.67872 4437.43544  1.93%   627   10s

Cutting planes:
  MIR: 2
  Zero half: 50

Explored 147 nodes (97808 simplex iterations) in 10.11 seconds (24.76 work units)
Thread count was 8 (of 8 available processors)

Solution count 6: 4524.68 4680.32 4738.86 ... 7775.44

Optimal solution found (tolerance 1.00e-04)
Best objective 4.524678717500e+03, best bound 4.524678717500e+03, gap 0.0000%
```

In [12]:
```python
# result
Minimum_total_time = model.objVal
print(f"Minimum total time: {Minimum_total_time:.2f} hours")
```

```
Minimum total time: 4524.68 hours
```

brief explanation

The updated objective function in model part 2 is the key distinction from the previous part 1 model. The new objective function is focusing on cut down on the overall time required to distribute milk across the entire state, rather than minimizing costs. This is accomplished by considering the distance between each distribution center and the LGAs they serve, divided by an assumed speed of 40 km/hr, and then adding 1.2 hours for each unit of milk demand in that specific LGA. The revision in the objective is a reflection of the importance placed on finishing the milk distribution process as quickly as possible to lower the chance of spoilage and guarantee on-time delivery.

2b

Multi-Objective Linear Programming (MOLP) is an efficient technique to accommodate the competing objectives of minimising cost and minimising delivery time for the milk distribution optimisation problem. The detailed steps will be listed below:

Step 1. Identify Decision Variables:
Binary variables representing the locations of distribution centers (x[i, j]), the number of tankers (t[i]), and other relevant parameters from the model will be considered for the decision variables.

Step 2. Identify Objectives:
Objective 1: Minimize Distribution Costs (Cost Objective) should be approximately = t1
Objective 2: Minimize Total Route Time (Time Objective) should be approximately = t2

Step 3. Identify Constraints:
Include constraints that represent capacity limitations, demand fulfillment, and any other relevant constraints in the milk distribution problem.

Step 4. Solve for Each Objective:
To find the optimal values for time and cost, I will solve the model separately for each of the two goals. This will provide me with two sets of optimized solutions, each for a different goal.

Step 5. Restate Objectives as Goals:
Set target values or goals based on the optimum values found for cost (t1) and time (t2).

Step 6. Create Deviation Functions:
Calculates the percentage or absolute amount by which a certain solution falls short of the objective.
Deviation Function 1 (Cost Deviation): Deviation1 = (Cost Objective - t1) / t1
Deviation Function 2 (Time Deviation): Deviation2 = (Time Objective - t2) / t2

Step 7. Assign Weights and Create Constraints:
Assign the deviation functions weights that correspond to the proportional value of cost and time. Set weight w1 for cost and weight w2 for time.
Create constraints that limit the weighted deviation functions.
w1 *Deviation1 <= Q*
w2 Deviation2 <= Q
Where Q is a variable that will be minimized to find a balanced solution.

Step 8. Minimize Q:
Utilizing the allocated weights and constraints, solve the modified problem with the goal of minimizing Q, which presents a trade-off between cost and time.

Step 9. Inspect and Adjust:
Examine the solution and determine if it meets the operational objectives. If required, rebalance the restrictions' weights and repeat the process until arrivgin at a workable solution that efficiently balances cost and time.

2c

Some of the additional goals or objectives might be important to consider:

Environmental Impact:
In the context of sustainability and corporate social responsibility in real-world, the environmental effect of a milk distribution problem is a important concern. It is imperative that the model include objectives pertaining to environmental effect in order to address this. Calculating the emissions related to the distribution fleet's fuel usage is one method to achieve. The model can evaluate the effects of several factors on emissions, including different vehicle kinds and routes. Additionally, the model may encourage environmentally beneficial behaviour by including goals to lower glasshouse gas emissions and lessen carbon footprints. This might entail modelling the use of alternative fuels, such electric or hybrid cars, and taking the corresponding decrease in emissions into account. For example setting additional objective function for emissions reduction = "obj_environment = gp.quicksum(emission_coefficient[i, j] *x[i, j]* vehicle_fuel_usage[i, j] for i in lgas_nsw for j in lgas_nsw if i != j)". Additional constraint to limit emissions = "model.addConstr(obj_environment <= max_emissions, name="Emissions_Limit")".

Economic Viability:
The sustainability and profitability of a milk distribution problem may also depend on maintaining economic viability. The model can solve this by including objectives that centre on economic variables. This involves taking into account the return on investment related to developing new distribution facilities or growing the network. The model should evaluate the financial viability of growth by accounting for the startup and ongoing expenses related to infrastructure modifications. It is possible to combine economic goals, including cost reduction, to make sure that infrastructure changes or development don't result in unaffordable cost rises. For example adding additional objective function for ROI and cost reduction = "obj_economic = gp.quicksum(roi[i] * n[i] - cost[i] for i in lgas_nsw)". Sensitivity analysis are a useful tool for assessing how different actions would affect economic viability financially. Through the integration of economic goals, the model contributes to the maintenance of the distribution network's long-term financial stability by coordinating sustainability and profitability.

In the code snippets above, I've included objectives for emissions reduction and economic viability. Balancing these objectives against the main objectives of reducing cost and distribution time requires trade-offs. The Pareto frontier, which illustrates the trade-off between multiple goals, can help find solutions that concurrently optimise several parameters. MOLP, which gives a systematic and organized way to navigate the complexities of multi-objective decision-making, may be used to assist in making informed and well-balanced options for this problem. The milk distribution model becomes a even more complete instrument for strategic decision-making, accounting for a wider variety of factors that effect the business, the environment, and many stakeholders by taking into consideration economic viability and environmental impact in real world. This strategy improves productivity while coordinating the distribution network with overarching objectives related to risk mitigation and sustainability.

In [ ]: