MESS-2020+1 Competition: Warehouse Location Problem with Store Incompatibilities — Problem description and rules —

Carmine Cerrone[1], Raffaele Cerulli[2], Sara Ceschia[3], Mario Pavone[4], Andrea Schaerf[3]

1. Università di Genova, carmine.cerrone@unige.it
2. Università di Salerno, raffaele@unisa.it
3. Università di Udine, {sara.ceschia,schaerf}@uniud.it
4. Università di Catania, mario.pavone@unict.it

June 15, 2021

1 Problem description

We propose the classical *Warehouse Location Problem* in which warehouses must be selected for opening and stores must be supplied by one or more open warehouses. The objective function to minimise is the sum of supply and opening costs. We consider the additional (hard) constraint that some pairs of stores cannot be supplied by the same warehouse.

1.1 Basic concepts

Warehouses: each warehouse has a *capacity* and a *fixed opening cost*.

Stores: each store has a *request* of quantity of goods to be completely satisfied by one or more warehouses.

Supply costs: it is given a matrix (Stores × Warehouses) that stores the cost *per unit* of transporting goods from each warehouse to each store.

Store incompatibilities: it is given a set of pairs of stores that cannot be served by the same warehouse.

1.2 Decision variables

The solver must select:

- for each pair $\langle s, w \rangle$, the quantity of goods moved from warehouse w to store s.
- warehouses to be opened

1.3 Constraints

The constraints are the following:

1. The total quantity of goods taken from a warehouse cannot exceed its capacity.
2. The total quantity of goods brought to a store must be exactly equal to its request.
3. Goods can be moved only from open warehouses.
4. Two incompatible stores cannot be supplied by the same warehouse.

1.4 Objective function

The objective function is the sum of the following two components:

- Cost of opening selected warehouses.
- Cost of supply the goods from the warehouses to the stores.

1.5 Instance files

Instances are provided in single file with the fixed structure that can be deduced from the following example.

```
Warehouses = 4;
Stores = 10;

Capacity = [100, 40, 60, 60];
FixedCost = [860, 350, 440, 580];
Goods = [12, 17, 5, 13, 20, 20, 17, 19, 11, 20];
SupplyCost = [|27, 66, 44, 55
              |53, 89, 68, 46
              |17, 40, 18, 61
              |20, 68, 44, 78
              |42, 89, 65, 78
              |57, 55, 49, 31
              |89, 101, 90, 16
              |37, 31, 23, 55
              |76, 60, 63, 44
              |82, 107, 91, 31|];

Incompatibilities = 3;
IncompatiblePairs = [| 1, 10 | 2, 7 | 8, 9 |];
```

The format is the MiniZinc data file, but the order of data is fixed in order to simplify the parsing with any languages (blanks are also fixed as in the example).

All values are positive integers. Incompatibilities are expressed by 1-based indexes of stores.

1.6 Solution files

Solutions can be written in two alternative file formats. The first is a matrix of supplied quantities, as exemplified by the following example.

```
[(0,0,0,12)
(0,0,0,17)
(0,0,0,5)
(0,0,0,13)
(0,0,20,0)
(0,7,0,13)
(0,16,1,0)
(0,0,19,0)
(0,11,0,0)
(0,0,20,0)]
```

Notice that the sum of each row is equal to the request of the store, and the sum of each column is less or equal to the capacity of the warehouse.

Warehouses that supply at least one store are implicitly considered open, the others are closed. In this solution, warehouses 2, 3, and 4 are open, warehouse 1 is closed.

The second format one is a list of triples, representing the indexes (1-based) of the store and the warehouse, and the quantity moved. The following example represents the same solution of the previous file written in the list format.

```
{(1,4,12), (2,4,17), (3,4,5), (4,4,13), (5,3,20), (6,2,7),
(6,4,13), (7,2,16), (7,3,1), (8,3,19), (9,2,11), (10,3,20)}
```

The solution provided above is feasible and has a total cost equal to 10842, composed by 9472 supply costs ($12\times55 + 17\times46 + 5\times61 + 13\times78 + 20\times65 + 7\times55 + 13\times31 + 16\times101 + 1\times90 + 19\times23 + 11\times60 +$t 20×91) and 1370 opening costs ($350 + 440 + 580$).

Note that this solution is not optimal. An optimal solution (cost 6757) is the following.

```
{(1,1,12), (2,1,17), (3,1,5), (4,1,13), (5,1,20), (6,1,8),
(6,4,12), (7,4,17), (8,1,19), (9,4,11), (10,4,20)}
```

Note also that the following solution (cost 7421) is infeasible, as warehouses 4 supplies incompatible stores 2 and 7.

```
{(1,3,12), (2,1,14), (2,4,3), (3,3,5), (4,3,13), (5,1,20),
(6,3,11), (6,4,9), (7,4,17), (8,3,19), (9,4,11), (10,4,20)}
```

1.7 Validator

The solution validator is provided as C++ source code and should be compiled using, for example, the GNU compiler g++. The solution received as command line parameters the instance and the solution files, like in the following example.

```
> ./WL_Validator.exe toy.dzn sol-toy.txt
```

It recognises the format in the solution file from the first character of the file (either `'['` or `'{'`).

The output of the validator for the solution provided above is the following.

```
Moving 12 goods from warehourse 4 to store 1, cost 12x55 = 660 (660)
Moving 17 goods from warehourse 4 to store 2, cost 17x46 = 782 (1442)
Moving 5 goods from warehourse 4 to store 3, cost 5x61 = 305 (1747)
Moving 13 goods from warehourse 4 to store 4, cost 13x78 = 1014 (2761)
Moving 20 goods from warehourse 3 to store 5, cost 20x65 = 1300 (4061)
Moving 7 goods from warehourse 2 to store 6, cost 7x55 = 385 (4446)
Moving 13 goods from warehourse 4 to store 6, cost 13x31 = 403 (4849)
Moving 16 goods from warehourse 2 to store 7, cost 16x101 = 1616 (6465)
Moving 1 goods from warehourse 3 to store 7, cost 1x90 = 90 (6555)
Moving 19 goods from warehourse 3 to store 8, cost 19x23 = 437 (6992)
Moving 11 goods from warehourse 2 to store 9, cost 11x60 = 660 (7652)
Moving 20 goods from warehourse 3 to store 10, cost 20x91 = 1820 (9472)
Opening warehouse 2, cost 350 (9822)
Opening warehouse 3, cost 440 (10262)
Opening warehouse 4, cost 580 (10842)
Number of violations: 0
Cost: 10842 = 9472 (supply cost) + 1370 (opening cost)
```

2 Competition rules

Rule 1: Participation is restricted to students of MESS 2020+1. Participants are allowed to work in groups of maximum 3 members.

Rule 2: Participants have to implement a metaheuristic technique to tackle the problem on a single thread. Hybrid techniques are also welcome. They can use any programming language that runs under Linux. The use of third-party software is allowed under the following restrictions: (*i*) it is free software, (*ii*) its behaviour is (reasonably) documented, (*iii*) it runs under Linux.

Rule 3: A dataset of 20 instances of different sizes are available from the opening day. A second dataset will be used to internally rank the top competitors (finalists). The instances belonging to the two datasets are called *Public* and *Hidden* instances, respectively. The Hidden instances will be released after the competition is closed.

Rule 4: The algorithms should take as input an instance file in the given format, and produce as output a solution in one of the described formats. The same version of the algorithm must be used for all instances.

Rule 5: The software should run for a maximum CPU time which depends on the size of the instance. We assume a reference CPU with 2.7GHz clock and 2GB of RAM, which is the one that will be used for the final ranking (see Rule 9 below).

For the reference machine the timeout in seconds s is a function of the number of warehouses w in the instance: $s = \lceil 10\sqrt{W} \rceil$. For example, for the instance `wlp-01.dzn`, with 50 warehouses, the timeout is 71s, and for instance `wlp-20.dzn`, with 3000 warehouses, the timeout is 548s.

For different CPU clock, the timeout must be adjusted proportionally.

Rule 6: The algorithm can be either deterministic or stochastic, but in both cases the results should be reproducible. In particular, the participants that use a stochastic algorithm should code their program in such a way that the exact run that produced each solution submitted can be repeated (by recording the random seed). They can try several runs to produce each submitted solution (each with the allowed computer time), but they must be able to repeat the run for any solution submitted.

Rule 7: Participants should submit for each Public instance the best score found by their algorithm within the timeout and the corresponding solution.

Rule 8: The set of 10 finalists will be selected after the competition deadline. Ranking of competitors will be based on the scores provided on the Public instances. The actual list will be based on the average of the ranks of scores on each single instance. The organisers reserve the right to increase the number of finalists, in case of a large participation.

Rule 9: The finalists will be given access to a virtual machine with Ubuntu Linux (with 2.7GHz clock and 2GB of RAM) on which they should install their code. The code will be run and tested by the organisers on the Hidden instances.

The solver submitted by the finalist should require as command-line arguments input and output file names, timeout in seconds, and, for stochastic solvers only, the random seed. For example (stochastic solver):

```
> ./my_solver.exe wlp-01.dzn sol-wlp01.txt 50 1542955064
```

Rule 10: Finalists' eventual place listings will be based on the ranks of 10 runs on each single hidden instance.

3 Important dates

Start: June 15th, 2021

Deadline: September 30th, 2021

Finalist announcement: October 10th, 2021

Software setup for finalists: October 20th, 2021

Final ranking announcement: November 10th, 2021

Paper due: December 10th, 2021

4 Prizes and publication

The top three of the competition ranking will receive a prize for the MESS 2020+1 organisers. All finalists will be invited to submit a manuscript of their work to be published in the special MESS 2020+1 Volume of the AIRO Springer Series.