Édition 2021

KI022, J.C. Alais, A. Parmentier and G. Dalle

1 Context

The so-called *Industrial Merchant* (IM) business line serves about 2 million clients and amounts to €9 billion out of the €20 billion total annual revenues of Air Liquide. In the IM business, the *Packaged Gases* activity (including filling plants and a distribution network for compressed gas) amounts to €3.4 billion, with around 600 plants in 80 countries. The plants and transportation that we are going to model for one country generate a large part of the costs, in addition to the raw material supplied by upstream liquid logistics known as *Bulk* activity.

More information about the Air Liquide group can be found in [1]

2 Problem description

A set I of clients i is distributed over a geographical area. The purpose of the problem we consider is to build the network of production and logistics centers that will enable to serve these clients at minimum cost. The period of optimization is one year.

We denote by S the set of possible sites where centers can be built. On each site s in S, a single center of any type (see below) can be built, and it is also possible to build no center at all on a site. In what follows, we will often use the letter s to refer to a center built on site s, assuming there is one.

Production centers produce gas cylinders[1], and send them either directly to the client, or to a distribution center. There are two kinds of production centers: plants and automated plants. Automated plants are more expensive to build, but they are also more cost-efficient during operation.

Distribution centers do not produce gas. A distribution center only receives cylinders from a single production center (not from another distribution center), and sends them to the clients.

Clients $i \in I$ can be served directly from a production center, or through a distribution

[1] bouteilles de gaz

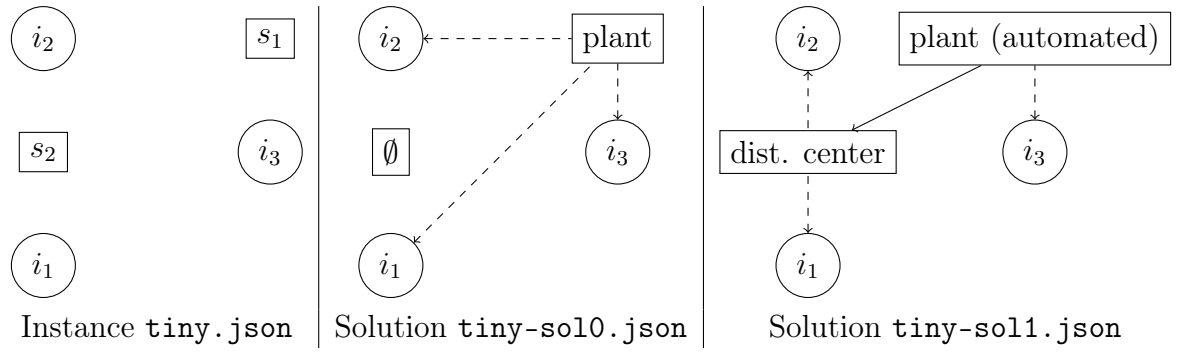

Figure 1: A tiny instance and two of its feasible solutions.
Plain arrows are primary routes, dashed arrows are secondary routes.

center. Each client i has a demand d_i which corresponds to the number of cylinders they consume in a year. They receive their whole demand from a single center.

2.1 Solution

The purpose of this hackathon is to make several decisions:

1. where to build production and distribution centers;
2. which production center serves each distribution center;
3. and which center (production or distribution) serves each client.

The goal is to serve all clients at minimum cost.

A solution x is a tuple $(P, D, (a_s)_{s \in P}, (p_s)_{s \in D}, (s_i)_{i \in I})$ where

- $P \subseteq S$ is the set of sites where we build production centers;
- $D \subseteq S$ is the set of sites where we build distribution centers;
- $a_s \in \{0, 1\}$ indicates, for a given production center $s \in P$, whether it is an automated plant ($a_s = 1$) or a standard plant ($a_s = 0$);
- p_s indicates which production center $p_s \in P$ serves a given distribution center $s \in D$ (the "parent" of s);
- s_i indicates which center $s_i \in P \cup D$ serves a given client $i \in I$ (the "parent" of i).

Two example solutions are represented on Figure 1.

Production center	Automation penalty	Distribution center
$c_P^{\mathrm{b}} = 800$ k€	$c_A^{\mathrm{b}} = 1200$ k€	$c_D^{\mathrm{b}} = 60$ k€

Table 1: Center opening costs

2.2 Constraints

The first constraint is that at most a single center can be built on each site. In other words,

$$P \text{ and } D \text{ are disjoint subsets of } S.$$

Furthermore, every distribution center must be served by a unique production center:

$$p_s \in P \text{ for all } s \in D.$$

Finally, every client must be served by a unique center (production or distribution):

$$s_i \in P \cup D \text{ for all } i \in I.$$

2.3 Objective

The objective is to minimize the sum of several different costs, detailed below.

Center building costs We pay a building cost for every center depending on its type. As announced, automatic plants are more costly to set up:

$$\texttt{building\_cost}(s, x) = \begin{cases} c_P^{\mathrm{b}} + a_s c_A^{\mathrm{b}} & \text{if } s \in P \\ c_D^{\mathrm{b}} & \text{if } s \in D \end{cases}$$

Table 1 provides the values of c^{b}.

Production costs We pay a unit production cost for each cylinder sent to a client i. It is associated with the filling and the handling processes (including picking and packing) at production and distribution centers. Since automated plants are more efficient than standard plants, their unit production cost is lower. If the cylinder goes through a distribution center, additional processes are triggered which make production more expensive. Of course, we have to multiply the unit cost by the number of cylinders sent (namely the demand d_i). We thus obtain the following total production cost for client i:

$$\texttt{production\_cost}(i, x) = \begin{cases} d_i(c_P^{\mathrm{p}} - a_{s_i} c_A^{\mathrm{p}}) & \text{if } s_i \in P \\ d_i(c_P^{\mathrm{p}} - a_{p_{s_i}} c_A^{\mathrm{p}} + c_D^{\mathrm{p}}) & \text{if } s_i \in D \end{cases}$$

Table 2 provides the values of c^{p}.

Production center	Automation bonus	Distribution center
$c_P^{\mathrm{p}} = 18$ €	$c_A^{\mathrm{p}} = 3.4$ €	$c_D^{\mathrm{p}} = 2$ €

Table 2: Unit production costs

Primary route	Secondary route
$c_1^{\mathrm{r}} = 0.015$ €/km	$c_2^{\mathrm{r}} = 3.63$ €/km

Table 3: Unit routing costs

Routing costs We pay a unit routing cost for each cylinder sent to a client i. The details of transportation (for instance the trucks used) depend on route types: primary routes link a production center to a distribution center, while secondary routes link any center to a client. Let $\Delta(s, s')$ denote the distance between sites s and s', and $\Delta(s, i)$ the distance between site s and client i. The unit cost of transporting one cylinder is a function of the length of the trip. If the site s_i serving client i is a production center, then we only pay for the trip $s_i \to i$ (secondary). But if s_i is a distribution center whose associated production center is p_{s_i}, then we pay for both trips $p_{s_i} \to s_i$ (primary) and $s_i \to i$ (secondary), which do not cost the same. Once again, we multiply this unit cost by the demand d_i and obtain:

$$\texttt{routing\_cost}(i, x) = \begin{cases} d_i \, [\quad 0 \qquad\quad + c_2^r \, \Delta(s_i, i)] & \text{if } s_i \in P \\ d_i \, [c_1^{\mathrm{r}} \, \Delta(p_{s_i}, s_i) + c_2^{\mathrm{r}} \, \Delta(s_i, i)] & \text{if } s_i \in D \end{cases}$$

Table 3 provides the values of c_1^{r} and c_2^{r}.

Capacity costs Production centers have a maximum capacity, which is higher for automated plants than for standard plants. Distribution centers on the other hand have no capacity constraint. Given a solution x, let us denote by $I(s)$ the set of clients for which production center $s \in P$ is involved in cylinder delivery (we do not define it for distribution centers since we do not need it). A production center s can serve client i directly (if $s_i = s$) or indirectly (if $p_{s_i} = s$), which means

$$I(s) = \{i \colon s_i = s\} \cup \{i \colon p_{s_i} = s\}.$$

Once the capacity u of a production center s is exceeded, we associate a highly prohibitive cost c^{u} to each additional cylinder served by this center. Let $[\cdot]^+$ denote the positive part $\max(\cdot, 0)$: this can be expressed as

$$\texttt{capacity\_cost}(s, x) = \begin{cases} c^{\mathrm{u}} \big[\sum_{i \in I(s)} d_i - (u_P + a_s u_A) \big]^+ & \text{if } s \in P \\ 0 & \text{if } s \in D \end{cases}$$

Table 4 provides the values of u.

Production center	Automation bonus
$u_P = 750$ k	$u_A = 225$ k

Table 4: Center capacities

Summary In conclusion, the cost of solution $x = (P, D, (a_s)_{s \in P}, (p_s)_{s \in D}, (s_i)_{i \in I})$ is given by the sum

$$\begin{aligned} \texttt{total\_cost}(x) = &\sum_{s \in S} [\texttt{building\_cost}(s, x) + \texttt{capacity\_cost}(s, x)] \\ &+ \sum_{i \in I} [\texttt{production\_cost}(i, x) + \texttt{routing\_cost}(i, x)] \end{aligned}$$

2.4 Problem

The goal of the problem is to find a feasible solution x of minimum cost $\texttt{total\_cost}(x)$.

3 Instance format and solutions

Instances are given under the `.json` format, which basically contains embedded dictionaries. In these dictionaries, the keys are always strings within quotation marks. Here is an example of a `tiny.json` instance illustrated on Figure 1

```
{
  "parameters": {
    "buildingCosts": {
      "productionCenter": 800000.0,
      "automationPenalty": 1200000.0,
      "distributionCenter": 60000.0
    },
    "productionCosts": {
      "productionCenter": 18.0,
      "automationBonus": 3.4,
      "distributionCenter": 2.0
    },
    "routingCosts": {
      "primary": 0.015,
      "secondary": 3.63
    },
    "capacityCost": 1000.0,
    "capacities": {
      "productionCenter": 750000,
      "automationBonus": 225000
    }
  },
```

```
  "clients": [
    {
      "id": 1,
      "demand": 7,
      "coordinates": [0.0, 0.0]
    },
    {
      "id": 2,
      "demand": 5,
      "coordinates": [0.0, 2.0]
    },
    {
      "id": 3,
      "demand": 12,
      "coordinates": [2.0, 1.0]
    }
  ],
  "sites": [
    {
      "id": 1,
      "coordinates": [2.0, 2.0]
    },
    {
      "id": 2,
      "coordinates": [0.0, 1.0]
    }
  ],
  "siteSiteDistances": [
    [0.0, 2.236],
    [2.236, 0.0]
  ],
  "siteClientDistances": [
    [2.828, 2.0, 1.0],
    [1.0, 1.0, 2.0]
  ]
}
```

The attribute `parameters` contains all the constants that have been introduced in the previous section:

- Attribute `buildingCosts` contains the building costs c^{b} summarized in Table 1, encoded as floats.
- Attribute `productionCosts` contains the production costs c^{p} summarized in Table 2, encoded as floats.
- Attributes `routingCosts` contains the routing costs c_1^{r} and c_2^{r} summarized in Table 3, encoded as floats.
- Attribute `capacityCost` contains the capacity cost c^{u}, encoded as a float.

- Attribute `capacities` contains the capacities u summarized in Table 4, encoded as integers.

The rest of the attributes contains the instance data.

- Attribute `clients` contains an array with the clients in I. Each client (element of the array) has three attributes:
  - An integer `id` corresponding to i, which is between 1 and $|I|$;
  - An integer `demand` that contains d_i;
  - Its latitude and longitude, encoded as floats. These coordinates do not appear in the definition of the problem. However, they may be used in heuristics.
- Attribute `sites` contains an array with the sites in S. Each site (element of the array) has two attributes:
  - An integer `id` corresponding to s, which is between 1 and $|S|$;
  - Its latitude and longitude, encoded as floats. These coordinates do not appear in the definition of the problem. However, they may be used in heuristics.
- Attribute `siteSiteDistances` contains the distance matrix $\Delta(s, s')$ between sites. Sites are enumerated by increasing `id`, and the distances are encoded as an array of lists of floats.
- Attribute `siteClientDistances` contains the distance matrix $\Delta(s, i)$ between sites (rows) and clients (columns). Clients and sites are enumerated by increasing `id`, and the distances are encoded as an array of lists of floats.

The expected solution files are also `.json` files, which must contain the following attributes:

- Attribute `productionCenters` contains an array whose elements have two attributes each:
  - An integer `id` corresponding to a site $s \in P$ where we build a production center;
  - An integer `automation` corresponding to a_s (0 or 1, not True or False!).
- Attribute `distributionCenters` contains an array whose elements have two attributes each:
  - An integer `id` corresponding to a site $s \in D$ where we build a distribution center;
  - An integer `parent` corresponding to the id of the production center $p_s \in P$ that serves distribution center s .
- Attribute `client` contains an array whose elements have two attributes each:

- An integer `id` corresponding to a client $i \in I$;
- An integer `parent` corresponding to the id of the center (production or distribution) $s_i \in P \cup D$ that serves client i.

Note that `clients` must contain every client id because every client needs a parent.

Figure 1 illustrates two solutions of this instance. The first one `tiny-sol0.json` has solution file

```
{
  "productionCenters": [{ "id": 1, "automation": 0 }],
  "distributionCenters": [],
  "clients": [
    { "id": 1, "parent": 1 },
    { "id": 2, "parent": 1 },
    { "id": 3, "parent": 1 }
  ]
}
```

while the second one `tiny-sol1.json` has the following solution file:

```
{
  "productionCenters": [{ "id": 1, "automation": 1 }],
  "distributionCenters": [{ "id": 2, "parent": 1 }],
  "clients": [
    { "id": 1, "parent": 2 },
    { "id": 2, "parent": 2 },
    { "id": 3, "parent": 1 }
  ]
}
```

In `tiny-sol0.json`, a single production center is built at s_1 and all the clients are served by it. This is a simple way of obtaining a feasible solution, even though it is not very interesting. It has cost `A: !!!`

In `tiny-sol1.json`, in addition to the production center at s_1 (which is now automated), we build an additional distribution center at s_2, by which clients i_1 and i_2 are now served. This second solution is more interesting and has cost `A: !!!`.

`G: Put reasonable coordinates and demands on the tiny instance so that the second solution is actually better than the first.`

`A: Update the parameters in the instance file and the solution costs`

4 How will the ranking be established?

!!! instances are provided:

- `A.json`

- `A: etc.`

The score of a team is the sum of the costs of the proposed solutions for each instance, without normalization (in other words, the large instances will weigh more and this is normal). The team with the best score, i.e. the lowest score, wins.

5 Some tips

You have to be very efficient to address such a problem in six hours. The team that is going to win is the one that manages to quickly produce decent solutions.

1. Divide the tasks
2. Immediately start coding the tools to parse an input file and create an output file
3. Keep it simple: it's not hard to build a feasible solution. Start by coding simple methods that give you pretty good solutions and avoid designing a very powerful algorithm that you will not be able to implement in 6 hours.

References

[1] "Universal registration document", Air Liquide; 2021

https://www.airliquide.com/sites/airliquide.com/files/2021/03/04/air-liquide-2020-universal-registration-document.pdf