Data in Brief 48 (2023) 109189

Contents lists available at ScienceDirect

Data in Brief

journal homepage: www.elsevier.com/locate/dib

Data Article

Real operational data for the concrete delivery problem[☆]

Alexandros Tzanetos[*], Maude Blondin

Multiobjective Optimization REsearch Lab (MORE Lab), Department of Electrical Engineering & Computer Engineering, Faculty of Engineering, Université de Sherbrooke, 2500, boulevard de l'Université Sherbrooke, Sherbrooke, J1K 2R1, Quebec, Canada

A R T I C L E   I N F O

Article history:
Received 31 March 2023
Revised 19 April 2023
Accepted 21 April 2023
Available online 29 April 2023

Keywords:
Concrete delivery problem
Ready-mixed concrete delivery problem
Operations research
Combinatorial optimization
Real data

A B S T R A C T

The data article describes a real operational dataset for the Concrete Delivery Problem (CDP). The dataset consists of 263 instances corresponding to daily orders of concrete from construction sites in Quebec, Canada. A concrete producer, i.e., a concrete-producing company that delivers concrete, provided the raw data. We cleaned the data by removing entries corresponding to non-complete orders. We processed these raw data to form instances useful for benchmarking optimization algorithms developed to solve the CDP. We also anonymized the published dataset by removing any client information and addresses corresponding to production or construction sites. The dataset is useful for researchers and practitioners studying the CDP. It can be processed to create artificial data for variations of the CDP. In its current form, the data contain information about intra-day orders. Thus, selected instances from the dataset are useful for CDP's dynamic aspect considering real-time orders.

[☆] Alexandros Tzanetos has a role as Section Editor of this journal but had no involvement in the peer-review of this article and has no access to information regarding its peer-review.

[*] Corresponding author.
E-mail address: Alexandros.Tzanetos@USherbrooke.ca (A. Tzanetos).

https://doi.org/10.1016/j.dib.2023.109189
2352-3409/© 2023 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

Specifications Table

Subject	Business, Management and decision sciences ➢ Transportation Management ➢ Management Science and Operations Research Engineering ➢ Industrial Engineering
Specific subject area	Management Science and Operations Research; Vehicle Routing Problems
Type of data	Table Structure Matrix
How the data were acquired	A concrete producer provided real operational data. The data consisted of details of daily orders, such as the requested concrete quantity, the expected delivery date, and the concrete mix type requested. Also, the travelling times between the instances' nodes were calculated using the Google API.
Data format	Raw Filtered MAT files MATLAB code
Description of data collection	The data correspond to real data parameters of 263 selected operational days of a concrete producer located in Quebec, Canada. We processed and cleaned the data to form problem instances for the CDP. The data also include some fixed parameters such as the unloading time on the construction site and the truck's cleaning time. These values are assumptions when data are incomplete or cannot be gathered.
Data source location	*City/Town/Region: Quebec* *Country: Canada*
Data accessibility	Repository name: Mendeley Data identification number: 10.17632/b78bc94d95.1 Direct URL to data: https://data.mendeley.com/datasets/b78bc94d95/1

Value of the Data

- This is the first available real-world data set for the CDP. Moreover, it contains significantly more data compared to the existing artificial datasets for the problem. In particular, it contains 263 instances of selected operational days of a concrete producer located in Quebec, Canada.
- Researchers developing optimization methods to solve the CDP can use this dataset for benchmarking. Also, practitioners can benefit from testing their software solutions on the various instances included in this dataset.
- The data can be further processed to create artificial data for variations of the CDP. Several formulations exist for the CDP, each representing a concrete producer's needs and constraints. Additional data are required when a formulation considers additional constraints of the problem. Researchers studying or proposing a new formulation can use part of the current data or extend them to validate their formulation.
- The data set is also useful for studying the occurrence of intra-day orders that represent a dynamic aspect of CDP.

1. Objective

The dataset described in the current article is the first available real-world data set for the CDP. Our recent systematic search and mapping review [1] pinpointed that no real-world data set is available. Even though data sets that correspond to typical operational days, such as [2], are used, they are not disclosed because they are under Non-Disclosure Agreements (NDAs).

Moreover, the largest artificial data set [3] contains 192 generated instances that do not correspond to real-world scenarios.

The scope of the collection of the data and publishing them anonymized is to amplify the research on the CDP. The usage of real instances will enable researchers from fields such as Operations Research, and Computer Science to develop more robust intelligent techniques to tackle the several aspects of the CDP. Moreover, researchers and practitioners can extend the current dataset. For example, they can approximate the value of the various fixed parameters of the dataset.

In the real-world setting, clients do not always order concrete in advance but also on the same day. The current data contain several instances with information about intra-day orders. Thus, researchers can use them to study a dynamic version of CDP, an aspect largely unexplored.

2. Data Description

We split this section into four subsections. The first subsection provides the necessary preliminaries for better comprehensibility of the data set. The second subsection describes the instances. The third subsection describes the complementary data, i.e., data related to the trucks and the concrete production sites, and the fixed parameters of the problem. Finally, the fourth subsection briefly describes the MATLAB scripts we provide alongside the data set. The variables' names are in the italicized font when they occur in the text, e.g., *MixCode*.

2.1. Preliminaries

The data correspond to the CDP which is a combinatorial optimization problem. In the literature, several models exist to describe this problem [1]. Among them, most models are based on graph-based interpretations of the problem, like the Vehicle Routing Problems (VRPs). The graph in such cases is defined as $G = (\mathcal{C}, \mathcal{D}, \mathcal{A})$, where $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ is the set of nodes representing the n construction sites to be served, $\mathcal{D} = \{D_1, D_2, \ldots, D_m\}$ denotes the set of m concrete production sites, and $A = (i, j | i, j \in \mathcal{CD})$ is the set of arcs, where the arc $\langle i, j \rangle$ denotes the vehicle travel from i-th node to j-th node. The graph of CDP is a network of production and construction sites. In a typical VRP, these sites are the depots and the customer nodes, respectively.

Below, we describe the main features of the problem:

- **Production Sites, i.e., Plants:** the concrete production sites of the network. To improve comprehensibility by practitioners, from this point further, we use the term Plants to refer to these nodes of the network. Plants are also the dispatching centers, meaning that all vehicles are assigned to one of the available plants.
- **Construction Sites:** the places where the concrete is delivered.
- **Orders:** every construction site is associated with one or more concrete orders. For each order, the client defines the requested volume of concrete, the time when the first delivery must arrive at the construction site, and the time lag between deliveries.
- **Demand:** the volume of requested concrete for a single order. All values are measured in m^3.
- **Deliveries:** usually, the requested concrete volume is significantly larger than the maximum trucks' capacity. Therefore, orders are divided into several deliveries.
- **Time lag between consecutive deliveries:** to ensure that each layer of concrete applied on the construction
- **Mix Type:** several mix types of concrete exist. Based on the construction type, clients request different concrete mix types.
- **Concrete delivery trucks:** several different capacity trucks used for concrete delivery exist. Dispatchers choose a truck according to the truck's capacity.

Each delivery operation constitutes a sequence of steps [4]:

1. the truck arrives at the Plant to load, or is already at the Plant if it is its Home Plant;
2. the driver places the truck at the loading deck;
3. the truck is loaded with the requested concrete mix;
4. the driver washes the exterior of the truck;
5. the driver transits the loaded truck to the construction site;
6. the driver parks the truck at the proper place; in some cases, a waiting time exists before this step, if no available unloading dock exists;
7. the concrete is unloaded;
8. the driver pulls away from the unloading place and washes the interior of the truck;
9. finally, the driver transits the empty truck to one Plant; usually, it is the Plant from where the truck started its trip.

The total operational time for a complete delivery considers all steps described above.

The following subsections describe the dataset content and the MATLAB script we provide for reproducibility.

2.2. Instances

We named the instances according to (a) the number of daily orders, and (b) the type of the CDP variation. i.e., static or dynamic. The CDP variation type comes after a unique serial number. To create this number, we grouped all instances of the same number of orders, and we put them in random order to avoid disclosure of any date information. Before the number of orders, we added the letter “o” for better comprehensibility. Then, “s” denotes the instances referring to the static CDP, and “d” denotes the dynamic CDP. For example, the instance named cdp_o2_4s denotes the fourth instance of two orders that correspond to the static CDP. Each instance file contains three .mat files:

1. **orders:** a table containing the details of each order for the specific day;
2. **instance_specs:** a structure containing specifications of the problem;
3. **prod2constr_matrix:** the travelling time matrix for the trips from Plants to Construction Sites;
4. **DistanceMatrix:** the corresponding instance’s distance matrix. We elaborate more on that in Section 3.6.

The ***orders*** table is a $n \times 9$ table, where the number of rows is equal to the n number of orders requested for the specific day. The columns correspond to the following variables; we denote the units in parenthesis, when applicable:

- ➢ **OrderDayID:** a sequential number used as the ID of the corresponding order. This number is defined in the interval $[1, n]$, where n denotes the number of orders requested for a specific day.
- ➢ **ClientDayID:** a sequential number used as the ID of the corresponding client. The same client may have placed several orders. In this case, this variable can be used for clustering the orders. This number is defined in the interval $[1, c]$, where c denotes the number of clients requested concrete for a specific day.
- ➢ **Usage:** the type of construction for which the concrete will be used for. This variable can be used to prioritize deliveries based on the construction type. The variable contains string values in French.
- ➢ **DefaultLoadSize (m^3):** the load per delivery requested by the client. If it equals zero, it means that the client has not defined the volume of load per delivery.
- ➢ **OrderedQuantity (m^3):** the volume of concrete requested by the client.
- ➢ **OrderedTime:** the time when the order was placed. This variable is given in the format hh:mm:ss. In instances corresponding to the static CDP, all values are equal to 00:00:00

denoting that all orders are known before the corresponding operational day. Otherwise, the time denotes the specific time that the order is placed within the day.

- **ExpectedDeliveryTime:** the time when the client requested to receive the concrete. This variable is given in the format hh:mm:ss.
- **LoadInterval** (*min*)**:** the requested interval between two consecutive deliveries.
- **MixCode:** a unique ID corresponding to different concrete mixes.

The *MixCode* variable denotes different concrete mixes. Each Mix ID corresponds to a specific mix type. However, to eliminate any information under NDA, we converted the *MixCode* to a pseudoID. i.e., a sequential number. This number is defined in the interval $[1, ct]$, where ct denotes the maximum number of available mix types in the current data set.

We provide a comprehensive description of the ***orders*** table's variables in Table 1, below.

Table 1
Description of the orders table.

Variable	Type	Data Type	Units
OrderDayID	Nominal	Integer	–
ClientDayID	Nominal	Integer	–
Usage	Nominal	String	–
DefaultLoadSize	Numeric	Float	m^3
OrderedQuantity	Numeric	Float	m^3
OrderedTime	Date	Datetime	–
ExpectedDeliveryTime	Date	Datetime	–
LoadInterval	Numeric	Integer	min
MixCode	Nominal	Integer	–

An important aspect of the CDP is the number of required deliveries for each order. The number of deliveries affects the solution to the problem. To calculate the maximum number of required deliveries for an order, three cases exist:

(a) **The client has defined a *DefaultLoadSize***, meaning they requested a specific concrete volume to arrive at each delivery. In this case, we use the following equation:

$$maxNofDeliveries_{C_1} = \left\lceil \frac{OrderedQuantity_{C_1}}{DefaultLoadSize_{C_1}} \right\rceil \tag{1}$$

where C_1 denotes the set of clients that have defined a DefaultLoadSize.

(b) **The client has not defined a *DefaultLoadSize*, and the *OrderedQuantity* exceeds the *DefaultQuantityPerTrip***. The *DefaultQuantityPerTrip* is a fixed concrete volume sent when the client does not specify the desired volume per delivery. In that case, the maximum number of deliveries is calculated as:

$$maxNofDeliveries_{C_2} = \left\lceil \frac{OrderedQuantity_{C_2}}{DefaultQuantityPerTrip} \right\rceil \tag{2}$$

where C_2 denotes the set of clients that have not defined a *DefaultLoadSize*, and their *OrderedQuantity* exceeds the *DefaultQuantityPerTrip*.

(c) **The client has not defined a *DefaultLoadSize*, but the *OrderedQuantity* does not exceed the *DefaultQuantityPerTrip***. In that case, the number of required deliveries is equal to one. Several trucks can perform the delivery. According to our industrial partner, it is a common practice to use a bigger capacity truck than required to fulfill such orders.

The structure ***instance_specs*** contains the following fields which provide information regarding the instance:

- **number_of_Orders:** the total number of orders on a corresponding day.
- **TotalConcreteQuantityOrdered:** the total ordered concrete quantity in m^3.

The matrix ***prod2constr_matrix*** corresponds to the submatrix P2C which is described in detail in Section 3.5. This submatrix contains the travelling times for the trips from Plants to Construction Sites.

2.3. Complementing Data

Apart from the instance files, we include data regarding the typical information of the problem, such as fleet data, Plants data, and fixed parameters corresponding to assumptions of the problem.

The file **Fleet_data.mat** contains the necessary information regarding the fleet of trucks. The file contains two tables:

- ➢ **AvailableTrucksPerPlant:** a 8 × 24 table containing the number of trucks of a specific capacity at each plant. The rows denote the eight (8) plants to which the data correspond. The 24 columns denote the different types of trucks' capacity.
- ➢ **ConcreteDeliveryTrucks:** this table contains the information for all available concrete delivery trucks in the fleet. Specifically, it includes three variables:
  1. **ID:** a sequential number used for each truck. This number is defined in the interval $[1, k]$, where k denotes the maximum number of concrete trucks in the fleet.
  2. **Capacity (m^3):** the capacity of the corresponding truck.
  3. **HomePlant:** the ID of the truck's Home Plant. This variable's values correspond to the pseudoID included in the Plants' tables, as explained below.

The file **Plants.mat** contains information regarding the concrete production sites. Specifically, it contains two fields:

- ➢ **LoadingRates:** a table with the mean loading rate at each Plant. The table contains two variables: (a) a pseudoID for each Plant, and (b) its mean loading rate. The pseudoID is a sequential number defined in the interval $[1, d]$, where d denotes the maximum number of Plants in the current data set.
- ➢ **plants_travelling_matrix:** the matrix consisting of the travelling times for the trips from Plant to Plant.

The **ProblemParameters.mat** is a structure containing the fixed parameters' values considered in case of missing data or inadequate information. For example, the pumps at each Construction Site may differ; thus, the unloading time varies. However, the clients do not provide information on the pump type at the Construction Site. Therefore, the unloading time at the Construction Site cannot be calculated. In this case, a fixed value exists, i.e., the *DefaultUnloadingTime*, which is 10 min. The values included in the structure are the following, where in parenthesis we denote the units, if applicable:

- ➢ **DefaultQuantityPerTrip (m^3):** in case the client did not specify the quantity of concrete to receive at each delivery, this fixed value denotes that the order is split into deliveries of 7.5 m^3 by default. We elaborate more on that, below.
- ➢ **DefaultLoadingTime (*min*):** it defines a default loading time at the Plant. This value can be used when no information regarding a Plant's loading rate exists.
- ➢ **DefaultPlantCleaningTime (*min*):** this parameter defines a default cleaning time at the Plant.
- ➢ **DefaultUnloadingTime (*min*):** the unloading time at the Construction Site depends on the on-site pump. However, the clients do not provide information on the pump type or its unloading rate. Thus, this parameter defines a default unloading time. Note also that another way to approximate the unloading time is the requested time lag between deliveries. According to our project's industrial partner, the clients specify the time lag between deliveries according to the required time to unload a truck. This practice ensures that the next truck will arrive while the previous one finishes unloading and thus, waiting times on-site are minimized.

- ➢ **CleaningTimeAtTheDefaultSite (*min*):** this parameter defines a default cleaning time at the Construction Site.
- ➢ **DriversPreparationTime (*min*):** this parameter denotes the default preparation time required for the driver before leaving the Plant.
- ➢ **GoogleTravelTimeMultiplier:** this is the coefficient to calculate the truck's travelling time between two locations. The travelling time between two locations given by Google API approximates the time required to travel by car. Thus, the corresponding time for a truck is calculated as:

$$tt_{ij}^{truck} = tt_{ij}^{car} \cdot GTTM \tag{3}$$

  where tt_{ij}^{truck} is the truck travelling time from location i to location j, tt_{ij}^{car} is the corresponding car travelling time, and GTTM denotes the *GoogleTravelTimeMultiplier*.
- ➢ **DriverBreakTime (*min*):** the driver's break time according to the local unions.
- ➢ **DriverMealTime (*min*):** the driver's mealtime according to the local unions.

According to our industrial partner, a predefined quantity to split the order into deliveries, i.e., the *DefaultQuantityPerTrip*, is associated with the capacity of the trucks used more often for concrete delivery. This type of truck has a capacity of 7.5 m^3. Indeed, looking at the Fleet_data.mat, we calculated that approximately 24% of the fleet corresponds to trucks with a capacity of 7.5 m^3. Note that in case the client did not specify the quantity of concrete to receive at each delivery and the order refers to a quantity less than 7.5 m^3, this default value stands as well. The reason is that it is common for concrete producers to not fully load trucks, according to the industrial partner. However, it is also possible to use a truck with a capacity less than 7.5 m^3 in case the client did not specify the quantity of concrete to receive at each delivery and the order refers to a quantity less than 7.5 m^3.

2.4. MATLAB Scripts

We provide several MATLAB scripts along with the data. We used these scripts for data processing. Some of our scripts refer to parts of the raw data that we cannot disclose. Therefore, in that case, we provide a snippet of the code in the relevant subsection of our article.

The provided MATLAB scripts are the following:

- ➢ **convertingDownloadedTimes2TravellingTimes:** which processes the raw times from an excel file corresponding to travelling times between Production Sites and Plants to convert them to minutes. More details are given in Section 3.5.
- ➢ **calculateNumberOfTrucksPerPlant:** which calculates the number of trucks of a specific capacity at each Plant. We describe this script's functionality at the end of Section 3.3.
- ➢ **convertTravellingTimeMatrix_2_DistanceMatrix:** which converts a given travelling time matrix to a distance matrix. We provide this script for researchers who study the CDP considering a distance matrix, as usual in Vehicle Routing Problems. Our dataset contains travelling times instead of distances. We explain the script's functionality in Section 3.6.

Apart from them, we used several MATLAB codes for processing the data. In the next section, we provide parts of these codes. Two reasons exist why we cannot provide the full codes presented in Section 3: (a) the original script includes codes that fall within the NDA, or (b) the code cannot be used without the raw data. In the second case, we provide the code for better comprehensibility of the methods performed to process the raw data.

3. Experimental Design, Materials and Methods

We formed the current dataset by processing the raw data provided by the industrial partner of the funded Mitacs project IT25421. Most of the raw data is sensitive and cannot be shared due to a Non-Disclosure Agreement (NDA). Therefore, we processed the given dataset and removed any information that could reveal sensitive information, such as the address of Plants and Construction Sites, the ID of clients, and the job number associated with each order.

After the removal of sensitive data, we removed any non-essential entries, such as incomplete entries. Then, we split the orders dataset into instances according to the expected delivery date.

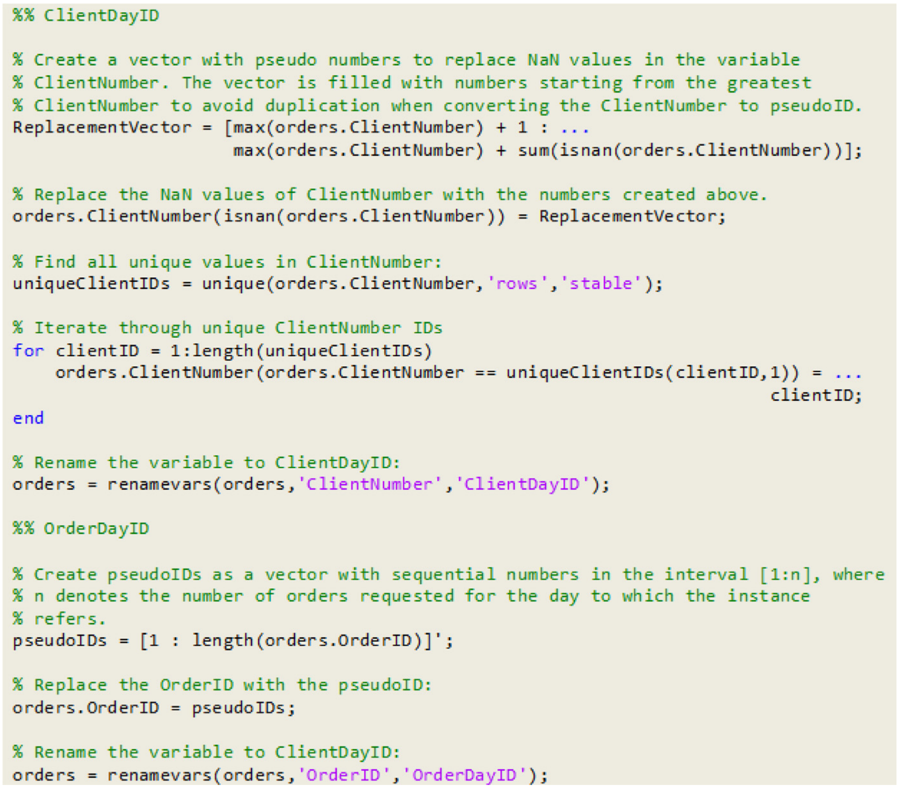
Regarding the Plants and Fleet data, we removed any entries not related to the problem. Specifically, we kept in the dataset only the concrete production Plants and the concrete delivery trucks, respectively. Concrete producers also own sites with other materials, such as lime and crushed stone. We excluded the information related to those sites from the data. Moreover, these companies own several types of trucks but not all of them deliver concrete. We also excluded the information related to non-concrete delivery trucks from the data.

3.1. ID-Oriented Variables

In general, any information that contains an ID number is sensitive and should be omitted from the public data set. Therefore, we created a pseudoID to replace ID-oriented variables.

3.1.1. Related to the Orders

The *OrderDayID*, the *ClientDayID*, and the *MixCode* are sensitive information that we cannot disclose. Therefore, we processed these variables by converting them to sequential numbers. For *OrderDayID* and *ClientDayID*, the sequential number corresponds to unique orders and clients of a specific day, respectively. Fig. 1 depicts the process followed for *OrderDayID* and *ClientDayID*.



```
%% ClientDayID

% Create a vector with pseudo numbers to replace NaN values in the variable
% ClientNumber. The vector is filled with numbers starting from the greatest
% ClientNumber to avoid duplication when converting the ClientNumber to pseudoID.
ReplacementVector = [max(orders.ClientNumber) + 1 : ...
                     max(orders.ClientNumber) + sum(isnan(orders.ClientNumber))];

% Replace the NaN values of ClientNumber with the numbers created above.
orders.ClientNumber(isnan(orders.ClientNumber)) = ReplacementVector;

% Find all unique values in ClientNumber:
uniqueClientIDs = unique(orders.ClientNumber,'rows','stable');

% Iterate through unique ClientNumber IDs
for clientID = 1:length(uniqueClientIDs)
    orders.ClientNumber(orders.ClientNumber == uniqueClientIDs(clientID,1)) = ...
                                                                      clientID;
end

% Rename the variable to ClientDayID:
orders = renamevars(orders,'ClientNumber','ClientDayID');

%% OrderDayID

% Create pseudoIDs as a vector with sequential numbers in the interval [1:n], where
% n denotes the number of orders requested for the day to which the instance
% refers.
pseudoIDs = [1 : length(orders.OrderID)]';

% Replace the OrderID with the pseudoID:
orders.OrderID = pseudoIDs;

% Rename the variable to ClientDayID:
orders = renamevars(orders,'OrderID','OrderDayID');
```

Fig. 1. MATLAB code to replace the *OrderDayID* and the *ClientDayID* with pseudo numbers.

The code's input is the instance's *orders* table. The output is the updated corresponding table where the *OrderID* and *ClientNumber* have been replaced by the pseudo numbers *OrderDayID* and *ClientDayID*, respectively.

For the *MixCode*, we used the full list of concrete mix codes to create a pseudocode. Fig. 2 shows part of the MATLAB code used to create a list of pseudo numbers corresponding to unique Mix Descriptions. As a first step, we process the concrete mix codes to replace the NaN values. We exclude entries that their *MixDescription* matches an existing *MixCode*. This step is not shown in Fig. 2 because it contains sensitive information. Then, we locate the unique *MixDescriptions* that are associated with NaN values of *MixCode* to replace them with some artificial *MixCodes*. After that, we follow a process similar to the one used for the *OrderDayID* and *ClientDayID*.

```
%% NaN Values

% Locate the NaN MixCodes in the list:
NaNMixCodes = MixCodesList(isnan(MixCodesList.MixCode),:);
% Find the unique MixCode descriptions in the NaN MixCodes list:
uniqueNaNMixCodes = unique(NaNMixCodes.MixDescription,'rows','stable');
% Create additional MixCodes to replace the NaN values:
additionalMixCodes = [max(MixCodesList.MixCode) + 1 : ...
                      max(MixCodesList.MixCode) + length(uniqueNaNMixCodes)]';

% Replace the NaN values with the corresponding additional MixCodes:
for replacement = 1:length(additionalMixCodes)
    MixCodesList.MixCode(strcmp(MixCodesList.MixDescription, ...
                         uniqueNaNMixCodes(replacement))) ...
                                                 = additionalMixCodes(replacement);
end

%% Unique MixCode IDs

% Find all unique MixCodes and the row where they first occur:
[uniqueMixCodes,firstoccurence,~] = unique(MixCodesList.MixCode,'rows','sorted');

% Create pseudoMixCodes as a vector with sequential numbers in the interval
% [1:mc], where mc denotes the maximum number of unique MixCodes.
pseudoMixCodes = [1 : length(uniqueMixCodes)]';

%% List of MixCode pseudoIDs

% Create the list of MixCode pseudoIDs:
pseudoMixCodeList = MixCodesList(firstoccurence,:);
% pseudoMixCodeList_actualID = MixCodesList(firstoccurence,:);

% Replace the actual ID with the pseudoID:
pseudoMixCodeList.MixCode = pseudoMixCodes;
```

Fig. 2. MATLAB code to create pseudo mix codes.

3.1.2. Related to the Fleet

The same applies to the trucks' IDs. We cannot disclose this information. Therefore, we created a sequential number to use as a pseudoID of the corresponding truck.

Furthermore, we replaced the actual Plants' ID with a pseudoID and we used them in the *HomePlant* variable denoting the trucks' Home Plant, i.e., home depot.

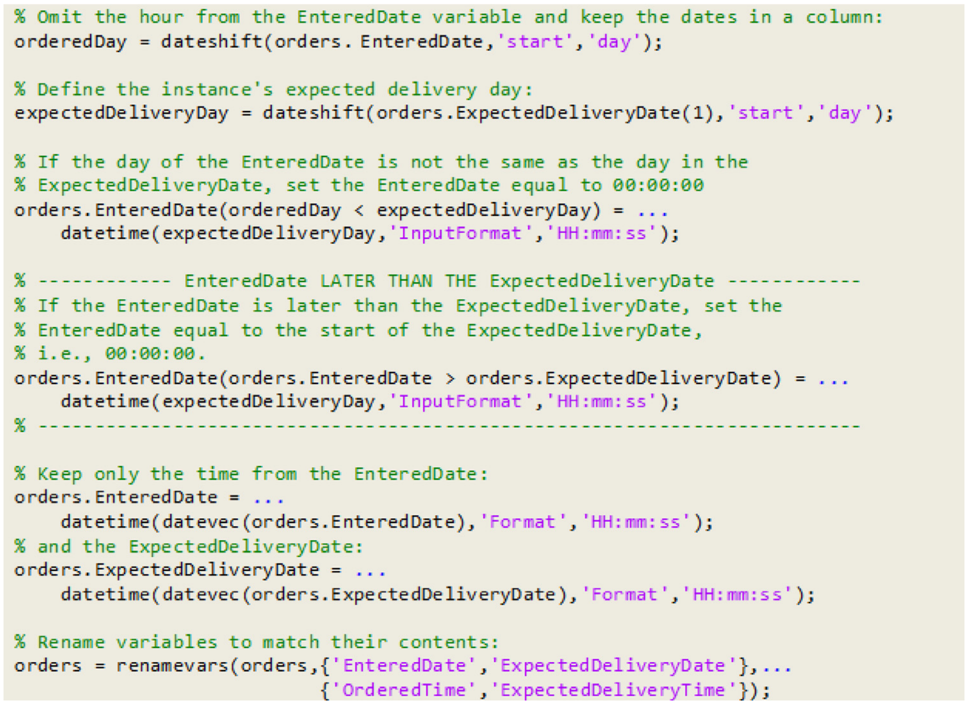
Details about both cases are provided in Section 3.3.

3.2. Ordered Date and Expected Delivery Date

Two variables in the raw *orders* table denote dates, i.e., *EnteredDate* and *ExpectedDeliveryDate*. The data of both variables matched the format dd-mmm-yyyy hh:mm:ss. However, the day, the month, and the year of the order fall within the NDA. Therefore, we omit this information from both variables, and the format of the outcome is hh:mm:ss, i.e., only the time information which is required for the problem.

Before applying this format transformation, we search for previous-day orders. If such orders exist, we replace their values with the start of the current day, i.e., the day to which the instance refers to. For example, if the *EnteredDate* is equal to ’13-Mar-2013 13:28:25′ and the *Expected-DeliveryDate* is equal to ’08-Apr-2013 12:30:00′, we replace the *EnteredDate* with ’08-Apr-2013 00:00:00′. In that way, we denote that the corresponding order is known at the beginning of the day.

Fig. 3 provides the MATLAB code used to process the two variables, as mentioned above. The input for this code is the *orders* table. The output is the same table, but the variables *EnteredDate* and *ExpectedDeliveryDate* have been processed and renamed to *OrderedTime* and *ExpectedDeliveryTime*, respectively.



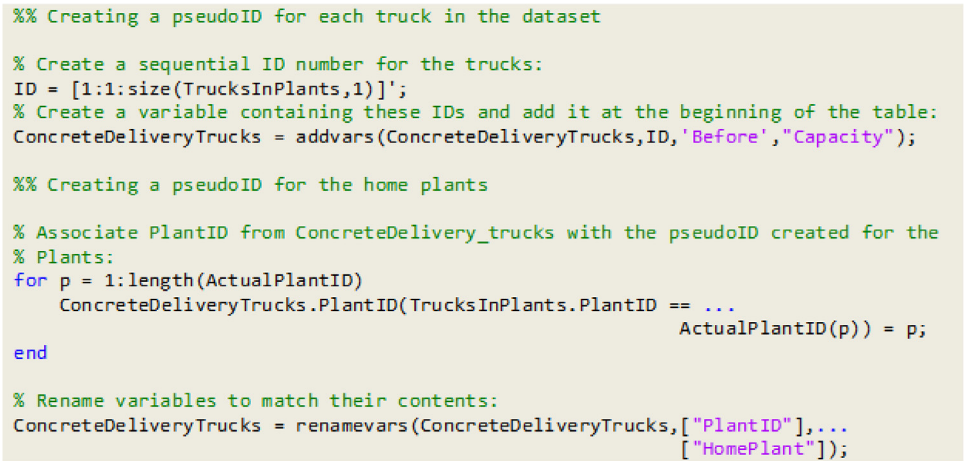
```
% Omit the hour from the EnteredDate variable and keep the dates in a column:
orderedDay = dateshift(orders. EnteredDate,'start','day');

% Define the instance's expected delivery day:
expectedDeliveryDay = dateshift(orders.ExpectedDeliveryDate(1),'start','day');

% If the day of the EnteredDate is not the same as the day in the
% ExpectedDeliveryDate, set the EnteredDate equal to 00:00:00
orders.EnteredDate(orderedDay < expectedDeliveryDay) = ...
    datetime(expectedDeliveryDay,'InputFormat','HH:mm:ss');

% ----------- EnteredDate LATER THAN THE ExpectedDeliveryDate ------------
% If the EnteredDate is later than the ExpectedDeliveryDate, set the
% EnteredDate equal to the start of the ExpectedDeliveryDate,
% i.e., 00:00:00.
orders.EnteredDate(orders.EnteredDate > orders.ExpectedDeliveryDate) = ...
    datetime(expectedDeliveryDay,'InputFormat','HH:mm:ss');
% -----------------------------------------------------------------------

% Keep only the time from the EnteredDate:
orders.EnteredDate = ...
    datetime(datevec(orders.EnteredDate),'Format','HH:mm:ss');
% and the ExpectedDeliveryDate:
orders.ExpectedDeliveryDate = ...
    datetime(datevec(orders.ExpectedDeliveryDate),'Format','HH:mm:ss');

% Rename variables to match their contents:
orders = renamevars(orders,{'EnteredDate','ExpectedDeliveryDate'},...
                          {'OrderedTime','ExpectedDeliveryTime'});
```

Fig. 3. MATLAB code for processing the *EnteredDate* and the *ExpectedDeliveryDate*.

Note that in the above code, we include a step where we manipulate the *EnteredDate* when it is later than the *ExpectedDeliveryDate* in the raw data. This inconsistency in the data does not occur very often but it is necessary to catch such errors in the raw data to ensure that the generated instances contain feasible solutions.

3.3. Fleet Information

Because we cannot disclose any information about the trucks owned by the industrial partner, we processed the available data to include as much important information as possible. After excluding sensitive information, we kept three variables: (a) the truck's ID, (b) the truck's capacity, and (c) the truck's home Plant. Then, we converted the truck's ID and the ID of its home Plant to pseudoIDs. Fig. 4 presents the part of the MATLAB script used to process the fleet data. *ActualPlantID* denotes a vector containing the actual Plants' ID from the raw data. We do not disclose the part of the code where we create the vector because the IDs contain sensitive data. The code's input is the raw fleet data included in the table *TrucksInPlants*, which are not provided. The output is the table *ConcreteDeliveryTrucks* that is included in the Fleet_data.mat of the current data set.



```matlab
%% Creating a pseudoID for each truck in the dataset

% Create a sequential ID number for the trucks:
ID = [1:1:size(TrucksInPlants,1)]';
% Create a variable containing these IDs and add it at the beginning of the table:
ConcreteDeliveryTrucks = addvars(ConcreteDeliveryTrucks,ID,'Before',"Capacity");

%% Creating a pseudoID for the home plants

% Associate PlantID from ConcreteDelivery_trucks with the pseudoID created for the
% Plants:
for p = 1:length(ActualPlantID)
    ConcreteDeliveryTrucks.PlantID(TrucksInPlants.PlantID == ...
                                                          ActualPlantID(p)) = p;
end

% Rename variables to match their contents:
ConcreteDeliveryTrucks = renamevars(ConcreteDeliveryTrucks,["PlantID"],...
                                                          ["HomePlant"]);
```

Fig. 4. MATLAB code for creating pseudoIDs for trucks and their Home Plants.

We also calculated the number of trucks of a specific capacity at each Plant. This matrix may be more useful for researchers and practitioners than the table *ConcreteDeliveryTrucks*. Indeed, instead of searching the table *ConcreteDeliveryTrucks* for available vehicles in a specific Plant, researchers can use a logical matrix based on the matrix *AvailableTrucksPerPlant*. The MATLAB script *calculateNumberOfTrucksPerPlant.m* calculates the number of trucks of a specific capacity at each Plant.

3.4. Plants' Loading Rates

The industrial partner provided us with the minimum and maximum estimated loading rate per Plant. We used those rates to calculate the mean loading rate as:

$$meanLoadingRate = \frac{maxLoadingRate - minLoadingRate}{2} \tag{4}$$

Note that each Plant has one loading dock. However, the dataset can be extended to study the Multi-dock Plant CDP, i.e., a case where more than one loading docks exist at the Plants. The research group which will perform such an extension, should collect data from concrete production companies and define the number of loading docks at each Plant. This number can be defined as a random number in the interval $[1, LD]$, where LD is the maximum number of loading docks that occurs in the collected data. Each dock can have the same mean loading rate calculated by Eq. (4).

3.5. Travelling Times Matrix

In VRPs, the distance or travelling matrix is essential because it denotes the travelling cost when traversing an arc of the problem's graph. In our dataset, we use the travelling matrix as the travelling cost from node i to node j. This is because one of the objectives considered in the industrial application we study is the minimization of total travelling time.

To collect the travelling times corresponding to the nodes of each instance, we used the Google API. At this point, we should note that the data correspond to a static interpretation of the CDP. Meaning that the travelling times are calculated once, and we assume this time does not change. Two options exist to solve the problem dynamically or consider external factors such as traffic. The first one is to consider travelling times as fuzzy sets. The second one is to download the travelling times in real time and update the matrix every time an incident occurs. In our case, we derived the travelling times calculated by Google API to create a comprehensive benchmark dataset containing real data. The travelling times matrix of each instance is composed of four submatrices, as shown below in Fig. 5.

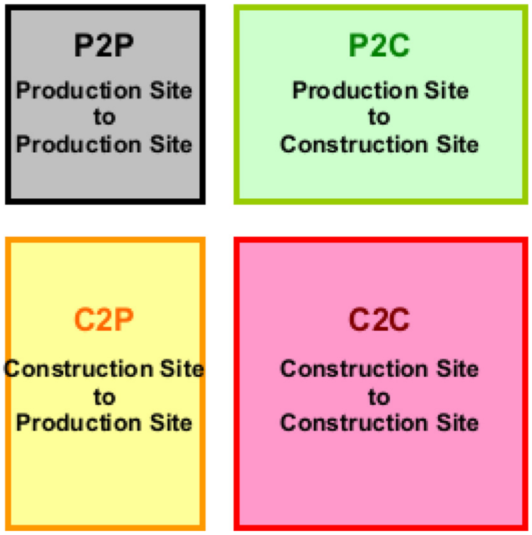

P2P
Production Site to Production Site
P2C
Production Site to Construction Site
C2P
Construction Site to Production Site
C2C
Construction Site to Construction Site

Fig. 5. Visualization of the submatrices composing the travelling times matrix for the problem.

The submatrix P2P contains fixed values because the distance between Production Sites, i.e., Plants, is fixed. Therefore, we calculated the travelling times once.

The submatrix C2P is the transposed matrix of the submatrix P2C because we consider a symmetric CDP, i.e., the travelling time from node i to node j is equal to the travelling time from node j to node i. Therefore, to decrease the number of requests to the Google API, we calculated only the P2C submatrix for each instance.

First, we appended the information of three variables from the raw data, i.e., *DeliveryStreet, DeliveryCity*, and *DeliveryState*, to form the complete address information for each order in the instance. The travelling times have been calculated with Google API, the service used by the industrial partner and their customers. We saved the outcome in Excel files for further processing. An example of the travelling time matrix outcome is given in Table 2.

Table 2
Example of travelling times outcome from Google API.

1 hour 35 mins	1 hour 16 mins	37 mins	47 mins	55 mins	1 hour 42 mins	25 mins	37 mins
1 hour 18 mins	1 hour 14 mins	11 mins	34 mins	53 mins	1 hour 37 mins	11 mins	11 mins
52 mins	33 mins	38 mins	16 mins	13 mins	59 mins	52 mins	38 mins
48 mins	34 mins	40 mins	28 mins	17 mins	1 hour 1 min	53 mins	40 mins
1 hour 23 mins	1 hour 13 mins	14 mins	36 mins	52 mins	1 hour 39 mins	8 mins	14 mins
48 mins	35 mins	41 mins	31 mins	20 mins	1 hour 1 min	54 mins	41 mins
1 hour 45 mins	1 hour 46 mins	46 mins	1 hour 10 mins	1 hour 26 mins	2 hours 4 mins	30 mins	46 mins

To convert the above matrix into a numerical matrix, we used the MATLAB script *convertingDownloadedTimes2TravellingTimes.m*. Initially, we remove the text “min”, or “mins”, and alter the “hour”, or “hours” into “:” to convert the cells in the format hh:mm. In case no hour value exists, e.g., “52 mins”, we append “0:” to match the format. Then, we apply a datetime format and calculate the travelling time in minutes as:

$$tt_{mins} = hh \times 60 + mm + {}^{ss}/_{60} \tag{5}$$

where *hh*, *mm*, and *ss* denote the hours, the minutes, and the seconds in the datetime, respectively. tt_{mins} denotes the outcome, i.e., the corresponding travelling time in minutes. The result of the above example can be seen in Table 3, below.

Table 3
Example of converted travelling times from Google API to numerical values (minutes).

95	76	37	47	55	102	25	37
78	74	11	34	53	97	11	11
52	33	38	16	13	59	52	38
48	34	40	28	17	61	53	40
83	73	14	36	52	99	8	14
48	35	41	31	20	61	54	41
105	106	46	70	86	124	30	46

Finally, the submatrix C2C denotes the travelling times between Construction Sites. These trips are prohibited in CDP. Therefore, this submatrix can be formed by zeros or infinite values according to the implementation of the algorithm used to solve the problem.

Once the final Travelling Times Matrix (TTM) is constructed, it should be multiplied by the parameter *GoogleTravelTimeMultiplier*:

$$TTM \leftarrow TTM \times GTTM \tag{6}$$

where GTTM denotes the *GoogleTravelTimeMultiplier*. Note that we use the assignment symbol $\leftarrow$ in Eq. (6) to show that the updated matrix replaces the initial one.

3.6. Distance Matrix

In addition to the travelling matrix, we also provide the corresponding distance matrix, calculated as the product of the travelling time matrix and the average truck's speed $\overline{velocity}_{truck}$. The calculation is described as:

$$d_{ij} = tt_{ij} \times \overline{velocity}_{truck} \tag{7}$$

where d_{ij} denotes the distance between nodes i and j, and tt_{ij} denotes the travelling time between nodes i and j.

The MATLAB script *convertTravellingTimeMatrix_2_DistanceMatrix.m* contains the complete process for the conversion from the travelling time matrix to the corresponding distance matrix. Initially, the script constructs the travelling time matrix according to the schema provided in Fig. 5. Note that in this case, multiplying with the *GoogleTravelTimeMultiplier* as described in

Eq. (6) is not required because we used an approximation related to regular vehicles, not concrete trucks. In the provided MATLAB script, we use the value 0.36 $^{km}/_{m}$ as the average truck's speed. However, we provide the option for the user to set a different average truck speed. We approximated the above value by calculating several average speed values as:

$$\bar{v}_{ij} = \frac{d_{ij}}{tt_{ij}} \tag{8}$$

Where $\bar{v}_{ij}$ denotes the calculated average vehicle speed, d_{ij} denotes the distance between nodes i and j, and tt_{ij} denotes the travelling time between nodes i and j. We collected several distances and travelling times from Google API to calculate multiple average speed values. Then, we computed the mean of these values to come up with the value used in our script.

Ethics Statements

This work meets the requirements of ethics as stated in (https://www.elsevier.com/journals/data-in-brief/2352-3409/guide-for-authors) and (https://www.elsevier.com/about/policies/publishing-ethics#Authors). This work also does not involve studies with animals and humans.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT Author Statement

Alexandros Tzanetos: Conceptualization, Methodology, Software, Validation, Investigation, Data curation, Visualization, Writing – original draft, Writing – review & editing; **Maude Blondin:** Supervision, Writing – review & editing.

Acknowledgments

This work was supported by Mitacs Project IT25421.

References

[1] A. Tzanetos, M. Blondin, Systematic search and mapping review of the concrete delivery problem (CDP): formulations, objectives, and data, Autom. Construct. 145 (2023) 104631, doi:10.1016/j.autcon.2022.104631.
[2] M. Maghrebi, T.S. Waller, C. Sammut, Scheduling concrete delivery problems by a robust meta heuristic method, in: 2013 European Modelling Symposium, 2013, pp. 375–380, doi:10.1109/EMS.2013.64.
[3] J. Kinable, T. Wauters, G. Vanden Berghe, The concrete delivery problem, Comput. Oper. Res. 48 (2014) 53–68, doi:10.1016/j.cor.2014.02.008.
[4] M. Durbin, K. Hoffman, OR PRACTICE—the dance of the thirty-ton trucks: dispatching and scheduling in a dynamic environment, Oper Res. 56 (2008) 3–19, doi:10.1287/opre.1070.0459.