A GRASP algorithm for the concrete delivery problem

Ousmane Ali[a], Jean-François Côté[a], Leandro C. Coelho[a,b]

[a] *CIRRELT, Université LAVAL, Canada*
[b] *Canada research chair in integrated logistics, Canada*

Abstract

This paper addresses a novel variant of the Concrete Delivery Problem (CDP), which involves the efficient scheduling of ready-mixed concrete deliveries to construction sites while balancing the conflicting goals of minimizing transportation costs and maximizing customer satisfaction. In this study, we propose an exact formulation and a heuristic approach based on the Greedy Randomized Adaptive Search Procedure (GRASP) to tackle this challenging CDP variant. This variant introduces realistic side constraints, including driver working shifts, a minimum driver working time, and overtime penalties. Additionally, it considers the case where customers may request multiple types of concrete delivered within the same time window. We assess the performance of our heuristic using new instances generated for this problem and provide a comparative analysis with another CDP variant from the literature to demonstrate its effectiveness.

Keywords: Vehicle scheduling, concrete delivery, GRASP, ready-mixed concrete

1. Introduction

Concrete is a widely used building material in construction projects. Its perishable nature is affected by many factors that impact its quality (Sinha et al., 2021), which is crucial for the durability and strength of the final construction. Concrete comes in two types: ready-mixed concrete (RMC) and site-mixed concrete (SMC). RMC is manufactured in a batch plant and delivered to the construction site, while SMC is produced on-site using raw materials stored on the construction site. Using SMC can avoid delays caused by road traffic, but it has a slower and more difficult production process, requires storage for mixing materials and equipment, and is suitable for low amounts of concrete. On the other hand, RMC has better quality and benefits from lower production costs (Muresan, 2019). However, to take advantage of these benefits, the

batch plant manager must ensure efficient and prompt delivery to the construction site, which may require a fleet of high-cost revolving drum trucks (concrete mixers) to dispatch the RMC.

Concrete delivery under the form of RMC is subject to many operational constraints that make the Concrete Delivery Problem (CDP) very challenging. In this paper, we study a variant of the CDP to schedule the daily production and dispatching of RMC for a company located in the province of Quebec, Canada. This company operates multiple batch plants with varying production rates, using a fleet of concrete mixers of different capacities. Each plant has its own fleet of trucks; however, under certain conditions, the trucks can move between plants if necessary. The trucks must return to their home plant at the end of the day. The company owns two types of trucks with different capacities and can call on an external fleet when needed. They serve construction sites from any of their plants, with the first delivery starting at the time specified by the customer. The loading and unloading of a concrete mixer depend on the truck capacity, the loading rate at the plant, and the unloading rate at the construction site. Drivers are allocated based on their daily work schedules. A customer may request several types of concrete to be delivered within the same time window, with no required sequence for the orders, but an order can only start after the completion of the previous order. This constraint generalizes the linked order constraints of Durbin and Hoffman (2008) where some customers place two orders for the same day and request that they are linked (the second order begins only after the first order is completed). The setting of our study is similar to a variant of the CDP previously studied in Schmid et al. (2009, 2010). However, we include the plant's production rate and driver shift schedule. The company uses a centralized dispatcher system to schedule all daily orders, but this system has issues satisfying all daily demands without using an external fleet or using different plants to serve the same order.

According to Blazewicz et al. (2019), the CDP combines vehicle routing with scheduling issues to plan routes to deliver concrete from batch plants (depots) to customers' construction sites. RMC is an on-demand product with a short life cycle from production through end use. It cannot be stored and cannot stay too long on a truck, or it will harden. Hence, concrete mixers must deliver RMC at the planned construction site shortly after its production. A customer quantity requirement is often greater than the truck size and must be fulfilled by multiple deliveries.

In that sense, CDP is similar to the vehicle routing problem with split delivery (Archetti and Speranza, 2008), except that the same truck may visit a customer more than once. Concrete hardens quickly, so multiple deliveries must be done back-to-back or at least close in time to avoid the problem of cold joint, which can reduce the strength and durability of the concrete. Customers request to be served within a specific time window, which can complicate truck-loading schedules when a plant can only load one truck at a time. Similarly, only one truck can unload at a time at a customer location, sometimes leading to concrete mixers queuing and waiting their turn to deliver. Furthermore, with trucks of varied sizes, loading, travel, and unloading times that may be uncertain, the CDP is a complex and challenging problem.

In this paper, we propose a mathematical model and a Greedy Randomized Search Procedure (GRASP) heuristic to solve a new variant of the CDP. Our model takes into consideration the working shifts of the drivers and the scheduling of multiple orders within the same time window at a construction site. To the best of our knowledge, this paper is the first that deals with these specific constraints in the context of the CDP.

The rest of the paper is organized as follows: Section 2 provides a literature review of previous works related to the CDP. Section 3 provides a formal description and mathematical model of the problem. Section 4 describes the GRASP algorithm and the constructive heuristics developed to solve this variant of the CDP. Section 5 presents computational experiments and sensitivity analyses to evaluate the proposed approach, and finally, we conclude in Section 6.

2. Literature review

Academic research on concrete batch and delivery began in the late 1990s. Tommelein and Li (1999) described RMC as a prototypical example of a JIT production system in construction and identified two practices for delivering it. One approach is for the customer to haul the product from the batch plant with their concrete mixer, while the other is for the batch plant to deliver the concrete directly to the customer's location. This latter approach is the one that has been studied in all related papers found in the literature. Several works to schedule and dispatch concrete production and delivery have mainly focused on simulation-related methods. These methods can be standalone (Zayed and Halpin, 2001) or can be hybridized with optimization

techniques, such as those used by Feng et al. (2004) and Lu and Lam (2005). Feng et al. (2004) used a combination of genetic algorithm (GA) and simulation process to minimize the total waiting time for trucks at a customer site. The study focused on loading homogeneous trucks at the same batch plant, with fixed loading and unloading durations. The GA was used to find the best loading sequence of RMC trucks to be assigned to different construction sites. The simulation process determined the loading, arrival, departure, and waiting time of trucks and thus evaluated the cost of each dispatching sequence. They evaluated their method using data from a batch plant in Taiwan with up to nine customers served. Lu and Lam (2005) used the same combination of GA and simulation to determine the optimal number of concrete mixers to be deployed and an optimal schedule for batching and delivering concrete. Their objective was to minimize the idle time of the site crew due to late concrete deliveries and truck queuing time. In this setting, it was also necessary to deliver a batch of mortar on-site to lubricate the unloading pump before the concrete delivery. As such, the simulation model also included the batch and delivery of mortar.

In addition to simulation-based methods, several other approaches have been used in the literature to solve the CDP. These include metaheuristics (Maghrebi et al., 2016; Yang et al., 2022), exact methods (Asbach et al., 2009; Kinable et al., 2014), matheuristics (Schmid et al., 2009, 2010), Benders decomposition (Maghrebi et al., 2014a), column generation (CG) (Maghrebi et al., 2014b), Lagrangian relaxation (Narayanan et al., 2015), and machine learning approaches (Graham et al., 2006). Matsatsinis (2004) designed a decision support system (DSS) for the dynamic routing of both concrete and pumps that may be necessary for some construction sites to aid in the unloading of concrete. The DSS considered the availability of three plants but stipulated that vehicles fulfilling the same order must all load at the same plant. Orders that could not be executed immediately could be postponed for the next day. The routing of the pumps was modeled as a multi-depot vehicle routing problem with time windows. Naso et al. (2007) proposed a sequential GA method combined with constructive heuristics to solve another variant of the CDP. In this problem, the plant's production schedule must consider orders that must be delivered to a customer's site as well as orders that must be picked up by the customer. The algorithm schedules the plant loading operations first before scheduling the truck deliveries.

The authors also developed a non-linear model that minimizes transportation costs, waiting times, outsourced costs, and overtime work. They ran experiments using real-world instances of a concrete supply chain in the Netherlands and found a reduction in the number of outsourced requests. Yan and Lai (2007) also considered overtime considerations in their paper, which focused on scheduling RMC for one batch plant with two loading docks. The study took into account that overtime wages are paid for factory and construction site operations after 4 PM. They developed a mixed-integer programming (MIP) model on a time-space network to minimize travel times and operating costs at both normal and overtime working hours at the plant and the construction sites. They tested the model using real data consisting of three days of operation using a two-stage algorithm. First, they solved the MIP relaxation with CPLEX. Then, they simplified the original model by fixing some decision variables before solving it. The algorithm was found to improve the actual plant operation by 10%. A time-space network is the key component of the real-time DSS developed by Durbin and Hoffman (2008) to solve a dynamic CDP every five minutes. The DSS can receive new orders, schedule them on the fly, and handle unexpected events such as plant closures, truck breakdowns, and delays in transportation times. The authors combined the DSS with a tabu search (TS) heuristic to warm start CPLEX, which made the model performant enough to solve instances with up to 1,500 loads per day with up to 250 trucks. The DSS also considers the case of a customer who places two orders, with the first being completed before the second starts. Further insights on the real-time planning and monitoring of CDP are available in Garza Cavazos (2021). Another variant of the CDP is modeled by Schmid et al. (2009) as an integer multicommodity network flow (MCNF) problem on a time-space network. In this paper, concrete is delivered using a heterogeneous fleet of vehicles, and each plant can load an unlimited number of trucks simultaneously. Some of the trucks have specialized equipment and must arrive first at certain construction sites to assist in unloading the concrete. The objective is to fulfill all orders, minimize the travel cost, and avoid delays between two consecutive unloading operations for an order. The model is typically solved using a matheuristic algorithm that combines the MCNF with a variable neighborhood search (VNS) heuristic. The method can quickly solve large problem instances with more than 60 orders per day. The same problem is addressed by Schmid et al. (2010), who proposed a

MIP model combined with a VNS and a very large neighborhood search (VLNS) to develop two matheuristics approaches. Comparisons between both matheuristics and a standalone VNS show that the former methods are much better and suitable for solving larger problem instances. These methods also provide better solutions for small to medium instances than the matheuristic used in Schmid et al. (2009). A pure VNS approach with the same problem but without the use of instrumentation has been applied by Payr and Schmid (2009).

Regarding objectives, most authors have focused on minimizing travel time and delays between consecutive deliveries. However, some authors have been more interested in maximizing customer satisfaction alone. We find these situations in the works of Durbin and Hoffman (2008) and Kinable et al. (2014). Kinable et al. (2014) introduce a general MIP and constraint programming (CP) models of the CDP reflecting the main constraints commonly found in all CDP works: time lag and no overlapping between consecutive deliveries, covering of all customers' demands, delivery time window, and heterogeneous fleet. However, the model did not include constraints limiting the time that concrete may stay on a truck. The authors propose a constructive heuristic that schedules the visits to the customers one by one according to the start time of the visit and the truck capacity. The procedure is invoked multiple times for different permutations of the customer's order which is determined using the steepest descent local search procedure. One of the paper's main contributions is the creation of the first public test instances for the CDP with up to 50 customers, four batch plants, and 20 concrete mixers. They found the CP model to be highly effective in finding high-quality solutions in a relatively short time or improving existing schedules, while the MIP model can be used to compute bounds, as it seems ineffective in solving large problem instances. Finally, the heuristic often yields good solutions in less than a second. A generalization of the MIP model of Kinable et al. (2014) is addressed in Asbach et al. (2009). This model simultaneously minimizes the total sum of travel costs and the penalty costs for customers with unfulfilled demand. A customer can request that all concrete deliveries come from the same plant or a subset of plants and that a delivery truck belongs to a subset of the vehicle fleet. The MIP model is used in a local search scheme as a black-box solver to reoptimize an incumbent solution in which a neighborhood operator has unfixed some variables. Tzanetos and Blondin (2023) provide an overview of the various methods used in the

literature to address the CDP and categorizes the problem formulations based on the different concepts used in the literature. They also discussed the consistency between industry needs and existing constraints and provided insights into the datasets corresponding to real-world cases, identifying the necessary data for practitioners.

3. Problem description

The focus of this paper is on the distribution of RMC from a Canadian company that operates in the greater Montreal area. When a customer places an order, it is received at a control center and immediately assigned to one of the company's batch plants. These plants produce the concrete and then deliver it to the customer. The problem involves a set of batch plants, a set of concrete-mixer drivers, and a set of customer orders.

Each of the company's plants has a single loading dock that can accommodate only one truck at a time. As a result, trucks often form a queue while waiting for their turn at the loading dock. Let B be the set of batch plants. The plants are heterogeneous, as each plant b has its hourly loading rate, represented by τ_b^l, which affects the duration of the loading process. After loading the concrete, the driver spends α_b minutes adjusting the concrete in the truck before heading to the customer site. Each plant has its own assigned fleet of trucks, but it can borrow trucks from other plants. Let l_j be the loading dock node associated with delivery node j. After loading RMC at l_j, it must be fully delivered to j at most before Δ time, which is the concrete lifetime. We denote L_b as the set of loading dock nodes of plant b and L as the set of all loading docks.

The company has two types of concrete mixer trucks with 8 and 12 cubic meters capacities. Each driver k is assigned to a particular batch plant and is responsible for driving a truck with capacity Q_k. The set of drivers is represented by $K = \bigcup K_b$, where K_b is the set of drivers scheduled to start their shift at batch plant b. A driver k is required to start his shift at h_k, work a minimum of μ hours and a maximum of η hours during regular working hours, with the possibility of overtime of up to Λ hours.

A driver typically loads RMC at his assigned batch plant but may be required to drive to and load at other plants if needed. The batch plant produces concrete on demand using recipes specific to each order. This means that a truck can only haul RMC for one order, even if there

is spare capacity. After unloading the RMC, the driver takes ρ minutes to clean the concrete mixer before proceeding.

A customer c requests one or more types of concrete to be delivered to their construction site on a specific day, with the delivery service starting at the due time a_c. The customer's unloading rate τ_c^u and the quantity to be unloaded give the time required to unload a truckload. We call an order o a request for a specific type of concrete. Let q_c be the sum of the demands q_o of each order o placed and a_c be the desired arrival time of the first concrete mixer. If an order requires more concrete than a single truck can carry, multiple deliveries are scheduled. To avoid cold joint problems with the concrete, subsequent deliveries of the same order must be made in close succession. We define a maximum time delay γ^1, after which no more deliveries are allowed for the same order. Let C be the set of customers, O_c be the set of all orders requested by customer c, and $O = \{O_c, c \in C\}$ be the set of all requested orders for all customers.

Each element of O_c must be entirely delivered before moving on to another order. The first order $o \in O_c$ to complete must have its first delivery start between the time windows $[a_c, a_c + \tau_c^w]$, while the subsequent orders can start their deliveries at most γ^2 time units after their preceding orders are completed. Here, τ_c^w is a user-defined parameter representing the time window duration, and γ^2 is the maximum time delay between the completion of an order and the start of the next order. A plant is assigned to an order, and it must supply all of its subsequent deliveries.

Let n_o be the number of deliveries needed to fulfill the order o. n_o is not known in advance because the trucks have different capacities. However, we can compute lower n_o^{min} and upper n_o^{max} bounds using the capacities of the largest Q_{max} and smallest Q_{min} available trucks.

$$n_o^{min} = \left\lceil \frac{q_o}{Q_{max}} \right\rceil \leq n_o \leq n_o^{max} = \left\lceil \frac{q_o}{Q_{min}} \right\rceil. \tag{1}$$

Let d_o^j be the j^{th} visit with load q_o^j for order o. We represent the fulfillment of order o by the visits to the ordered set of delivery nodes $D_o = \left(d_o^1, d_o^2, \cdots, d_o^{n_o}\right)$. The deliveries of customer c are the ordered set $D_c = (D_{o_1}, D_{o_2}, \cdots, D_{o_{|O_c|}})$, where o_r is the r^{th} delivered order. We will refer to $d \in D_c$ ($d \in D_o$) as the d^{th} potential delivery of customer c (order o). $D = \bigcup_{c \in C} D_c$ is the union of all delivery nodes.

A solution to the problem involves decisions about truck loading schedules, driver assignments to different deliveries, and truck arrival times at construction sites for unloading. For a batch plant, the decision involves choosing which driver to load, when to load them, how much to load, and which construction site to deliver. For a driver, the decision is to determine the sequence of loading depots and delivery sites. For a construction site, the decision involves determining the arrival times of all scheduled deliveries for the day.

Each driver leaves and returns to their home plant every day. We represent the home plant of a driver k with a starting depot s_k and an ending depot e_k. Let S and E be the sets of starting and ending depots, respectively.

We define our problem on a directed graph where $V = \{S \cup L \cup D \cup E\}$ is the set of nodes. The arc sets are $A = \{(i,j) \mid i,j \in V\}$, $A^D = \{(i,j) \mid i,j \in D\}$, and $A^L = \{(i,j) \mid i,j \in L\}$. A corresponds to allowed movements of drivers from node i to node j. For each driver k, the allowed movements are the following:

- From the starting depot s_k to a loading dock $l \in L$ or to the ending depot e_k.
- From a loading dock $l \in L$ to a delivery node $d \in D$.
- From a delivery node $d \in D$ to a loading dock $l \in L$ or to the ending depot e_k.

For a customer c, arcs in A^D link consecutive delivery nodes of the same order $\{(i,j) \in D_o, o \in O_c, i < j\}$, and pair of delivery nodes of two different orders $\{(i, d^0_{o_2}), i \in D_{o_1}, i \geq n^{min}_{o_1}, o_1, o_2 \in O_c, o_1 \neq o_2\}$. Arcs in A^L link all pairs of loading docks of the same batch plant.

We define $\delta^+(i) = \{(i,j) \in A\}$ and $\delta^-(i) = \{(j,i) \in A\}$ as the outcoming and incoming arc sets of node $i \in V$; $\delta^+_D(i) = \{(i,j) \in A^D\}$ and $\delta^-_D(i) = \{(j,i) \in A^D\}$ are the outcoming and incoming arc sets of delivery node $i \in D$. Similarly, $\delta^+_L(i) = \{(i,j) \in A^L\}$ and $\delta^-_L(i) = \{(j,i) \in A^L\}$ are the outcoming and incoming arc sets of loading node $i \in L$. t_{ij} is the time to travel from i to j.

Let the binary variable x^k_{ij} be 1 if driver k travels from node i to j. Binary variable y_o is 1 when order o is completely served; v_i and w_i are the start and end of the loading (unloading) operation at node $i \in L \cup D$. Binary variable u_{ij} is 1 if node j is served just after i, the service being either an unloading or a loading operation. Variable q^k_j is the quantity to be loaded towards

j with vehicle k. Let w_k^1 be a continuous variable indicating the difference between the driver's work time and the minimum number of hours to be worked in a day, and w_k^2 indicating the difference between the driver's work time and the normal work time. Let g_i be the time between the due date and the first service start for customer i.

The objective function minimizes total travel cost (TC), penalties associated with unfulfilled orders, first delivery delays (FDD), driver underutilization costs (DUC), and driver overtime costs (DOC). Together, these components drive the optimization process to find a solution that efficiently balances travel costs, customer satisfaction, on-time delivery, driver utilization, and scheduling constraints.

$$\min \sum_{k\in K}\sum_{(i,j)\in A} t_{ij}x_{ij}^k + \beta_1\sum_{o\in O}(1-y_o) + \beta_2\sum_{i\in C} g_i + \sum_{k\in K}\left(\beta_3 w_k^1 + \beta_4 w_k^2\right) \tag{2}$$

$$\sum_{j\in\delta^+(i)} x_{ij}^k = 1 \qquad i\in S, k\in K \tag{3}$$

$$\sum_{j\in\delta^-(i)} x_{ji}^k = 1 \qquad i\in E, k\in K \tag{4}$$

$$v_j \geq w_i - M\left(1-x_{ij}^k\right) \qquad i\in S, j\in\delta^+(i), k\in K \tag{5}$$

$$v_j \geq w_i + \alpha_b + t_{ij} - M\left(1-x_{ij}^k\right) \qquad b\in B, i\in L_b, j\in\delta^+(i), k\in K \tag{6}$$

$$v_j \geq w_i + \rho + t_{ij} - M\left(1-x_{ij}^k\right) \qquad i\in D, j\in\delta^+(i), k\in K \tag{7}$$

$$w_i \geq v_i + \frac{q_j^k}{\tau_b^l} - M\left(1-x_{ij}^k\right) \qquad b\in B, i\in L_b, j\in\delta^+(i), k\in K \tag{8}$$

$$w_j \geq v_j + \frac{q_j^k}{\tau_c^u} - M\left(1-x_{ij}^k\right) \qquad c\in C, j\in D_c, i\in\delta^-(j), k\in K \tag{9}$$

$$w_j \leq v_i + \Delta + M\left(1-x_{ij}^k\right) \qquad j\in D, i\in\delta^-(j), k\in K \tag{10}$$

$$v_{d_o^0} \geq a_c \qquad c\in C, o\in O_c \tag{11}$$

$$g_c \geq v_{d_{o_1}^0} - a_c - M\left(\sum_{j\in\delta_D^-(d_{o_1}^0)} u_{jd_{o_1}^0}\right) \qquad c\in C, o_1\in O_c \tag{12}$$

$$g_c - a_c \leq \tau_c^w \qquad c\in C \tag{13}$$

$$v_{d_{o_1}^0} \geq w_j - M\left(1-u_{jd_{o_1}^0}\right) \qquad o_1\in O_i, j\in\delta_D^-(d_{o_1}^0) \tag{14}$$

$$v_{d_{o_1}^0} \leq w_j + \gamma^2 + M\left(1-u_{j,d_{o_1}^0}\right) \qquad o_1\in O, j\in\delta_D^-(d_{o_1}^0) \tag{15}$$

$$\sum_{o_1\in O_c}\sum_{j\in\delta_D^-(d_{o_1}^0)} u_{j,d_{o_1}^0} = |O_c|-1 \qquad c\in C, |O_c|>1 \tag{16}$$

$$\sum_{o_1 \in O_c} \sum_{j \in \delta_D^+(d_{o_1}^0)} u_{d_{o_1}^0, j} = |O_c| - 1 \qquad c \in C, |O_c| > 1 \tag{17}$$

$$\sum_{j \in \delta_D^+(d_{o_1}^0)} u_{d_{o_1}^0, j} \leq 1 \qquad o_1 \in O \tag{18}$$

$$\sum_{j \in \delta_D^-(d_{o_1}^0)} u_{j, d_{o_1}^0} \leq 1 \qquad o_1 \in O \tag{19}$$

$$\sum_{j \in \delta_D^+(d_{o_1}^0)} u_{d_{o_1}^0, j} + \sum_{j \in \delta_D^-(d_{o_1}^0)} u_{j, d_{o_1}^0} \geq 1 \qquad o_1 \in O \tag{20}$$

$$v_{j+1} \geq w_j - M\left(1 - u_{j,j+1}\right) \qquad o \in O, j \in D_o, j \geq 1 \tag{21}$$

$$v_{j+1} \leq w_j + \gamma^1 + M\left(1 - u_{j,j+1}\right) \qquad c \in C, o \in O_c, j \in D_o, j \geq 1 \tag{22}$$

$$u_{j,j+1} \geq u_{j+1,j+2} \qquad o_1 \in O, j \in D_{o_1}, 1 \leq j \leq n_{o_1} - 2 \tag{23}$$

$$u_{j,j+1} \geq \sum_{l \in L} x_{lj} \qquad o_1 \in O, j \in D_{o_1}, j \geq 1 \tag{24}$$

$$v_j \geq w_i - M\left(1 - u_{i,j}\right) \qquad i \in L, j \in \delta_L^+(i) \tag{25}$$

$$\sum_{j \in \delta_L^+(i)} u_{i,j} \leq 1 \qquad i \in L \tag{26}$$

$$\sum_{j \in \delta_L^-(i)} u_{j,i} \leq 1 \qquad i \in L \tag{27}$$

$$\sum_{j \in \delta_L^-(i)} u_{j,i} \geq \sum_{k \in K} \sum_{j \in \delta^-(i)} x_{ji}^k \qquad i \in L \tag{28}$$

$$\sum_{j \in \delta_L^-(i)} u_{j,i} \geq \sum_{j \in \delta_L^-(i+1)} u_{j,i+1} \qquad i \in L \tag{29}$$

$$\sum_{j \in \delta_L^-(i)} u_{j,i} + \sum_{j \in \delta_L^+(i)} u_{i,j} \geq \sum_{k \in K} \sum_{j \in \delta^-(i)} x_{ji}^k \qquad i \in L \tag{30}$$

$$\sum_{k \in K} \sum_{j \in D_o} q_j^k = q_o \qquad o \in O \tag{31}$$

$$q_j^k \leq \sum_{i \in \delta^-(j)} Q^k x_{ij}^k \qquad j \in D, k \in K \tag{32}$$

$$\sum_{k \in K} \sum_{j \in \delta^+(i)} x_{ij}^k \leq 1 \qquad i \in L \cup D \tag{33}$$

$$\sum_{k \in K} \sum_{j \in \delta^+(i)} x_{ij}^k = \sum_{k \in K} \sum_{j \in \delta^-(i)} x_{ji}^k \qquad i \in L \cup D \tag{34}$$

$$w_k^1 \geq \mu + h_k - v_{e_k} \qquad k \in K \tag{35}$$

$$w_k^2 \geq (v_{e_k} - h_k) - \eta \qquad k \in K \tag{36}$$

$$w_{e_k} \leq h_k + \Lambda \qquad k \in K \tag{37}$$

$$0 \leq q_j^k \leq Q^k \qquad j \in D, k \in K \tag{38}$$

$$x_{ij}^k \in \{0,1\} \qquad (i,j) \in A, k \in K \tag{39}$$

$$u_{ij} \in \{0,1\} \qquad (i,j) \in A^D \tag{40}$$

$$y_o \in \{0,1\} \qquad o \in O \tag{41}$$

$$v_i \geq 0, w_i \geq 0 \qquad i \in V \tag{42}$$

$$w_k^1 \geq 0, w_k^2 \geq 0 \qquad i \in V \tag{43}$$

$$g_c \geq 0 \qquad c \in C. \tag{44}$$

β_1 to β_4 are the penalty coefficients of each component of the objective function (2). Constraints (3) and (4) state that driver k leaves his start node exactly once a day and returns to his end node. A driver cannot serve a node before the start of his shift with constraints (5). Constraints (6) and (7) set a driver to take some time after loading or unloading to adjust the concrete or clean the truck before moving on to the next node. The duration of the loading operation depends on the plant's loading rate and the amount q_j^k of RMC loaded (8). Similarly, the unloading service depends on the site's rate and q_j^k (9). Unloading operations must end at most Δ minutes after loading begins (10).

Constraints (11) to (15) ensure that the first service of any customer c must start after the due time a_c. This first service may be performed at the first delivery node of any order, and one order must be completed before another is started. It also enforces the precedence constraints between the last delivery of an order and the first delivery of the following order. Constraints (16) to (20) find the delivery sequence of all orders. With constraints (21)–(24), two trucks cannot unload at the same time for consecutive deliveries of the same order. Additionally, these constraints impose a maximum time delay between the two trucks. Similarly, constraints (25)–(30) ensure that two trucks cannot be loaded at the same time at a plant. Constraints (31) require that the cumulative load of all concrete mixers serving an order must equal the required quantities, and (32) bound a driver load. Constraints (33) require a driver to only visit a loading/delivery node once. Constraints (34) are degree constraints. Constraints (35)–(36) calculate the difference between a driver's hours of service and the minimum and normal hours of service. Finally,

constraints (37) to (44) define the nature and bounds of the variables.

Figure 1 presents a solution for an instance. The instance has two plants, nine drivers, and five customers, with two customers having multiple orders. Customer 4 has two orders (o_4–o_5) assigned to plant 1, and customer 5 has three orders (o_6–o_8) assigned to plant 2. All the order requirements are in the Table 1. Figure 1 provides a visual representation using a Gantt chart of the loading and unloading operations performed by the drivers at each plant. The light gray, dark gray and black colors represent a customer's first, second and third order. We can see that the loading operations for each plant do not overlap, and the unloading operations occur sequentially, one after the other, at each worksite. At plant 1, loading operations for $C3$ are performed first by drivers $3, 5, 4, 2$ followed by loadings for $C1$. The second order of $C5$ is delivered, followed by the second, and finally, the first order. All first deliveries start at their due time, except for $C2$, which is 20 minutes late. There is a delay between each delivery of $C2$. However, it is less than the maximal time delay of 25 minutes defined for this example. We can see that some drivers load at both plants. For example, driver 5 loads at plant 1 to perform the second delivery of $C3$ then drives to plant 2 for the first delivery of $C2$. Driver 1 is originally assigned to plant 1 but starts his first service at 2 before returning to his home plant. The loadings at plant 2 show how the loading dock is alternately assigned for orders of customers $2, 4$ and 5 between 06:25 and 10:00. We obtain this solution using a heuristic solution approach that we describe next.

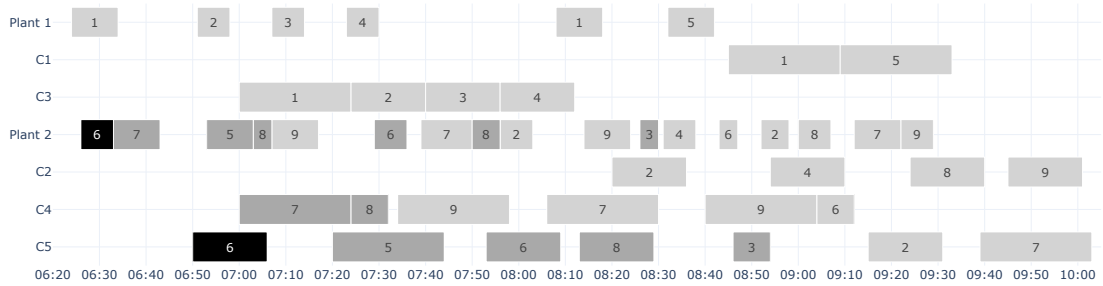
Table 1: Information of the illustration instance

Order	1	2	3	4	5	6	7	8
Customer	1	2	3	4	4	5	5	5
Demand	24	32	36	40	16	20	32	8
Plant	1	2	1	2	2	2	2	2
TW	08:45	08:00	07:00	07:00	07:00	06:50	06:50	06:50

4. Constructive heuristics and GRASP

This section presents the heuristic approach to solving the Concrete Delivery Problem. Our method consists of constructing feasible solutions of the CDP with randomized greedy heuristics

Figure 1: Gantt chart of the loading and unloading operations of the illustration instance



and iteratively invoking these heuristics in the GRASP metaheuristic. From now on, we represent the scheduling of a delivery node d as the pair (L_d^k, U_d^k) of loading and unloading tasks. $L_d^k = (b, q_d, v_b, w_b)$ indicates that driver k will load q_d m^3 of RMC at plant b between timeslots $[v_b, w_b]$ to serve d. $U_d^k = (k, v_d, w_d)$ indicates that driver k will start unloading at time v_d and finish at time w_d. We then call a solution $Sol = \left\{ \cup_{1 \leq j \leq |D|} (L_j^k, U_j^k), k \in K \right\}$ the set of all pairs of loading and unloading tasks performed.

4.1. GRASP algorithm

The Greedy Randomized Search Procedure (GRASP) algorithm is a multi-start metaheuristic for combinatorial optimization problems proposed by Feo and Resende (1989). It has been successfully applied to solving scheduling problems (Bamoumen et al., 2023; Yepes-Borrero et al., 2023) and transportation-related problems such as the prisoner transportation problem (Ferone et al., 2023) and the truck and trailer routing problem (Villegas et al., 2011). The GRASP is a two-phase iterative sequence of constructive and local search algorithms. A feasible solution is generated in the construction phase using a greedy randomized algorithm. Then, a local search phase tries to improve it. Algorithm 1 outlines the GRASP algorithm. We start with a list of greedy randomized heuristics and iterate on each to construct an incumbent solution Sol on which we apply a local search procedure to explore its neighborhood. The best solution found so far is then stored. To avoid redundant computations, a hash table T ensures that the local search is not applied twice to the same solution. The procedure stops and returns the best overall

solution when certain stopping conditions are met, such as reaching a time limit or a maximum number of iterations.

Algorithm 1 Pseudo-code of the GRASP algorithm

Input: H: List of constructive greedy randomized heuristics

```
1  Sol* ← ∅    Cost(Sol*) ← ∞
2  Initialize an empty hash table T ← ∅
3  while conditions not met do
4  |  for each heuristic ∈ H
5  |  |  Sol ← heuristic()
6  |  |  if Sol ∉ T then
7  |  |  |  Sol ← LocalSearch(Sol)
8  |  |  |  Add Sol to T
9  |  |  |  if Cost(Sol) < Cost(Sol*) then
10 |  |  |  |  Sol* ← Sol
11 return S*
```

For the construction phase, we developed two greedy randomized heuristics: the customer-based insertion with backtracking, which schedules customers one at a time, and the delivery-based insertion.

4.2. Customer-based insertion with backtracking heuristic

The customer-based insertion heuristic builds a solution from scratch by iteratively scheduling one customer at a time. When a customer is selected, all deliveries needed to serve that customer are scheduled before moving on to the next customer. As shown in Algorithm 2, between lines 4 and 38, we iterate over a shuffled list of orders for each customer.

To schedule an order o, we begin with its first delivery node. Once this node is scheduled, we proceed to schedule the subsequent delivery nodes until o is completely served. For each delivery node d, the algorithm identifies between lines 9 and 12 each driver capable of visiting this node using the $SimulateVisit$ procedure. For all drivers able to serve d, at line 12 we store the details about the pair of loading and unloading tasks and its cost in $M[d]$.

If at least one driver can serve the node (line 13), we randomly select at line 16 a pair of loading and unloading tasks whose cost falls within a specified range based on a parameter θ and insert this pair into Sol at line 18. Then we update the order remaining demand and proceed with the next delivery node at line 19.

In cases where no driver can visit the node d, the algorithm employs a backtracking scheme to address two distinct scenarios. First, if the visit to d cannot be scheduled due to cold-joint constraints (i.e., the time between consecutive deliveries exceeds γ^1), the $DelayPreviousDelivery$ procedure is employed. This procedure determines if a previous delivery $D_o^{j'}$ of the same order exists such that d can be successfully scheduled in the current solution by delaying the start of service of $D_o^{j'}$ by λ. If $D_o^{j'}$ exists, the schedules of all delivery nodes between $D_o^{j'}$ and d are removed, and the scheduling of order o resumes from $D_o^{j'}$.

The procedure $DelayPreviousDelivery$ computes the difference λ between the earliest time a driver can visit D_o^j and the end of service of its immediate predecessor node D_o^{j-1}. Then, it tries to find one of the precedent nodes of D_o^j for which we can delay the start of service of at least $\lambda = min(\lambda, \gamma^1)$. If such a node exists, the procedure returns it and the value of the delay. Then we backtrack to this node and restart the scheduling process.

In cases where infeasibility arises from other violated constraints, the algorithm attempts to backtrack to a precedent node and retry the scheduling with another driver. At line 31, the algorithm searches for the first precedent node $D_o^{j'}$ with remaining pair of tasks in $M[D_o^{j'}]$. If found, the schedules of all delivery nodes between $D_o^{j'}$ and d are removed, and backtracking occurs to $D_o^{j'}$. In this iteration, since $M[D_o^{j'}]$ is not empty, the $SimulateVisit$ procedure is not called again; instead the stored elements of $M[D_o^{j'}]$ are utilized.

If no precedent node suitable for backtracking is found, it indicates that the current customer cannot be scheduled. Consequently, all previously scheduled visits to its delivery nodes are removed before advancing to the next customer (as described in lines 23–25 and lines 32–34). The procedure stops when all customers are evaluated and returns the constructed solution Sol at the end.

The $SimulateVisit$ procedure takes a partial solution, a delivery node d, a driver k and the delay value λ as parameters. Its goal is to find the earliest time when the loading service for d can start at its home plant b. When planning to visit delivery node d of order o and customer c with driver k, we first compute his delivery time v_d as follows. If d is the first visited node of customer c, then we expect its service to start at $v_d = a_c + \lambda$. Otherwise, if the previous node j visited before d belongs to the same order o, then the service at d may begin at the random

value $w_j + rand(\lambda, \gamma^1)$ to respect the allowed time between two consecutive deliveries of the same order. We randomize v_d to allow more diversity in our solution space. Finally, if j and d belong to different orders, then $v_d = w_j + rand(0, \gamma^2)$.

Let $\bar{q}_o$ be the remaining demand of order o. Driver k can supply d with the quantity $q_d^k = \min(\bar{q}_o, Q^k)$. We determine the loading start time v_l of the corresponding dock node l by subtracting the travel time, adjustment time, and loading duration from v_d: $w_l = v_d - t_{dl} - \alpha_b - q_d^k/\tau^l{}_b$. If j is the last node where k was unloading with a service duration s_j, then $w_l = \max(w_l, v_j + s_j + t_{jl})$. Now, we have the earliest time k can start loading at plant b, but if the loading dock is occupied at that time, k will have to wait. To find the real loading time, we use a data structure that stores all timeslots when a plant is occupied. Then, starting from w_l, we search forward for the first timeslot with duration $q_d^k/\tau^l{}_b$ where we can insert this loading operation and update w_l accordingly. Changing w_l can cause violations of the time window or cold-joint constraints, especially for an order's subsequent deliveries, and thus make the schedule infeasible. So with this new value of w_l, we recompute v_l and check for constraint violations.

Algorithm 2 is designed with a list of customers as input. Initial experimentation has revealed that the sequence of this list affects the final solution. Within the GRASP framework, we invoke the customer-based heuristic multiple times with various customer orderings. These include sorting by decreasing demand and time window, sorting by increasing demand and time window, and finally, with a randomized list. When confronted with scenarios where several customers require their services to start simultaneously or within the same time window, we observed that with Algorithm 2, some customers remain unscheduled when resources are limited. To address this limitation, we introduce the delivery-based insertion heuristic, which allows each customer to have one of its delivery nodes considered for scheduling at each iteration.

4.3. Delivery-based insertion heuristic

The delivery-based insertion heuristic (Algorithm 3) iteratively constructs a solution from scratch by scheduling one delivery node at a time. The algorithm starts with an empty solution Sol and initially fills a candidate list CL with the first node of a randomly selected order for each customer. Within each iteration of the *while* loop, the algorithm filters CL to obtain a reduced candidate list CL' of deliveries based on a specified criterion, such as prioritizing deliveries with

Algorithm 2 Customer-based insertion algorithm

Input: Sol: empty solution, K: vehicles set, Π: list of customers

1 $M \leftarrow \emptyset$ // List of pairs of loading - unloading tasks for each delivery node
2 **for each** $c \in \Pi$
3 Shuffle O_c
4 **for each** $o \in O_c$
5 $\lambda \leftarrow 0$, $\bar{q}_o \leftarrow q_o$, $j \leftarrow 1$
6 **while** $\bar{q}_o > 0$ **do**
7 $d \leftarrow D_o^j$
8 **if** *(*$M[d] = \emptyset$*) or (*$\lambda > 0$*)* **then**
9 **for each** $k \in K$
10 $(L_d^k, U_d^k, cost, f) \leftarrow$ SimulateVisit(Sol,d,k,λ)
11 **if** *f=1* **then**
12 $M[d] \leftarrow M[d] \cup \{(L_d^k, U_d^k, cost, k)\}$
13 **if** $M[d] \neq \emptyset$ **then**
14 $c_{min} \leftarrow \min\{cost \mid (L_d^k, U_d^k, cost, k) \in M[d]\}$
15 $c_{max} \leftarrow \max\{cost \mid (L_d^k, U_d^k, cost, k) \in M[d]\}$
16 Randomly choose $(L_d^k, U_d^k, cost, k)$ in $M[d]$ such that $cost \leq c_{min} + \theta(c_{max} - c_{min})$
17 $M[d] \leftarrow M[d] \backslash \{(L_d^k, U_d^k, cost, k)\}$
18 $Sol \leftarrow Sol \cup \{(L_d^k, U_d^k)\}$
19 $\bar{q}_o \leftarrow \bar{q}_o - q_d^k$, $\lambda \leftarrow 0$, $j \leftarrow j + 1$
20 **else**
21 **if** *cold-joint constraints* **then**
22 $(D_o^{j'}, delay) = DelayPreviousDelivery(Sol,d)$ // $j' < j$
23 **if** *no* $D_o^{j'}$ *exists* **then**
24 $Sol \leftarrow Sol \backslash \{(L_i^k, U_i^k), i \in D_c\}$
25 Go to line 2 // continue with next customer
26 $Sol \leftarrow Sol \backslash \{(L_i^k, U_i^k), i \in (D_o^{j'}, .., D_o^{j-1})\}$
27 Update $\bar{q}_o$
28 $j \leftarrow j'$, $\lambda \leftarrow delay$
29 Go to line 6 // backtrack and retry to insert $D_o^{j'}$
30 **else**
31 Search for a precedent delivery node $D_o^{j'}, j' < j$ with $M[D_o^{j'}] \neq \emptyset$
32 **if** *no* $D_o^{j'}$ *exists* **then**
33 $Sol \leftarrow Sol \backslash \{(L_i^k, U_i^k), i \in D_c\}$
34 Go to line 2 // continue with next customer
35 $Sol \leftarrow Sol \backslash \{(L_i^k, U_i^k), i \in (D_o^{j'}, .., D_o^{j-1})\}$
36 Update $\bar{q}_o$
37 $j \leftarrow j'$
38 Go to line 6 // backtrack to precedent delivery node $D_o^{j'}$
39 **return** Sol

the highest (lowest) demands, earliest (latest) due dates, or lowest order numbers. This filtering is performed by the *Filter* procedure, allowing the algorithm to explore different neighborhoods. Thus, within the GRASP framework, we can invoke the delivery-based heuristic multiple times

with different filtering procedures which evaluate different neighborhoods.

Between lines 14 and 18, for each delivery d in CL', the algorithm determines the best available driver k who can visit it. Then, the corresponding pair of loading and unloading tasks (L_d^k, U_d^k), along with its costs, are stored in $M[d]$. This is done using the $SimulateVisit$ procedure described earlier. Then, the algorithm constructs at line 20 a list T containing all pairs of loading and unloading tasks for the delivery nodes in CL'. If T is empty, the algorithm assumes that the customers associated with the nodes in CL' cannot be scheduled, and it removes those customers' existing schedules from the solution. It then excludes CL' from CL at line 25 and proceeds to the next iteration.

Otherwise, from line 26 to line 29, the algorithm randomly selects from T a pair of loading and unloading tasks whose cost falls within a specified range determined by the parameter θ and inserts this pair into Sol. The algorithm then updates the remaining orders and customer quantities and excludes the scheduled delivery node from CL. If an order o is not completely fulfilled after visiting d_o^j, the next delivery node d_o^{j+1} is added to CL. Otherwise, if the customer has remaining orders, the first delivery node $d_{o'}^1$ of another randomly selected order o' is added to CL.

4.4. Local Search

After the construction phase, the local search phase invokes local search operators to either repair or improve the incumbent. One operator is used to repair a solution where not all customers are scheduled. Then four operators are used to improve the incumbent using a first-improvement strategy. The local search continues until no more improvements can be made or a timeout is reached.

4.4.1. Repair operator

The Remove and Reschedule operator is used to address situations where a solution contains unscheduled customers. This operator first selects an unscheduled customer c from the solution and identifies a set Π_c containing scheduled customers sharing the same time slot with c. The schedules of all elements of Π_c are then removed from the solution and retried to be scheduled again along with c.

Algorithm 3 Delivery-based insertion algorithm

Input: *Sol*: empty solution, Π: list of customers, *Filter*: filtering procedure

$CL \leftarrow \emptyset$
$M \leftarrow \emptyset$ `// List of pairs of loading - unloading tasks for each delivery node`
$\bar{q} \leftarrow \emptyset$ `// List of remaining demands`
for each $c \in \Pi$
  $\bar{q}_c \leftarrow q_c$
  Select a random order $o \in O_c$
  $CL \leftarrow CL \cup \{d_o^1\}$
  $\bar{q}_o \leftarrow q_o$
while $CL \neq \emptyset$ **do**
  $CL' \leftarrow Filter(CL)$
  $T \leftarrow \emptyset$ `// List of loading and unloading tasks`
  for each $d \in CL'$
    $best\_cost \leftarrow +\infty$
    for each $k \in K$
      $(L_d^k, U_d^k, cost, f) \leftarrow$ SimulateVisit(Sol,d,k,λ)
      if *f=1 and* $cost < best\_cost$ **then**
        $M[d] \leftarrow \{(L_d^k, U_d^k, cost, k)\}$
        $best\_cost \leftarrow cost$
    if $M[d] \neq \emptyset$ **then**
      $T \leftarrow T \cup M[d]$
  if $T = \emptyset$ **then**
    for each $d \in CL'$
      $c \leftarrow$ customer associated with d
      $Sol \leftarrow Sol \backslash \{(L_j^k, U_j^k), j \in D_c\}$
    $CL \leftarrow CL \backslash CL'$
  else
    $c_{min} \leftarrow \min\{cost \mid (L_d^k, U_d^k, cost, k) \in T\}$, $c_{max} \leftarrow \max\{cost \mid (L_d^k, U_d^k, cost, k) \in T\}$
    Randomly choose $(L_{d_o^j}^k, U_{d_o^j}^k, cost, k) \in T$ such that $cost \leq c_{min} + \theta(c_{max} - c_{min})$
    $Sol \leftarrow Sol \cup \{(L_{d_o^j}^k, U_{d_o^j}^k)\}$
    $\bar{q}_o \leftarrow \bar{q}_o - q_{d_o^j}^k$, $\bar{q}_c \leftarrow \bar{q}_c - q_{d_o^j}^k$ // $o \in O_c$
    $CL \leftarrow CL \backslash \{d_o^j\}$
    if $\bar{q}_o \neq 0$ **then**
      $CL \leftarrow CL \cup \{d_o^{j+1}\}$
    else if $\bar{q}_c \neq 0$ **then**
      Select another order $o' \in O_c$
      $CL \leftarrow CL \cup \{d_{o'}^1\}$
return *Sol*

4.4.2. Improvement operators

The first improvement or Load Backward operator is used when dealing with solutions containing customers whose first delivery is late. In such cases, the goal is to schedule the loading start time for the first delivery node for these customers as early as possible. This operator is

implemented by iterating backward through the timeslots data structure when calculating the loading start time within the $SimulateVisit$ procedure. By doing so, the operator aims to find a loading start time that allows for earlier delivery, thus reducing the late first delivery cost. The other improvement operators are based on the swap and relocate moves.

We have implemented two moves based on the swap operator. The Swap Driver operator explores alternative driver assignments by swapping drivers assigned to two delivery nodes. This allows for the assessment of different driver configurations, potentially leading to improvements in solution quality. The Swap Load operator exchanges loads between a delivery node and the last delivery node of the same order. In our greedy randomized heuristics, all deliveries of an order are typically filled to capacity, except for the last one. This operator investigates the effect of delivering this last load much earlier in the delivery sequence. Delivering the final load earlier in the sequence can improve the loading operations, as it may be easier to find an empty timeslot for inserting a loading task with a smaller load.

We also implemented two moves based on the relocate operator. The Consolidate Load operator is applied on an order with a remaining demand between $]Q_{min}, Q_{max}]$. If this remaining demand is served with two drivers with capacity Q_{min}, then the operator tries to consolidate it to be served by a single driver with a capacity of Q_{max}. Next, the Relocate Driver operator removes a delivery node from its current route and inserts it in another driver's route. This enables the redistribution of delivery nodes among drivers, potentially reducing drivers underutilization and overtime costs.

5. Computational experiments

In this section, we present and discuss the results of computational experiments conducted using the GRASP heuristic described in Section 4. The implementation of the GRASP algorithm is coded in C++. We run our experiments on the benchmark data used in Kinable et al. (2014), and on a new dataset extracted from delivery operations records provided by our industry partner.

5.1. Generation of instances

The dataset used in this section was obtained from our partner and contains records representing delivery operations from up to 8 plants for 35 days. From this historical data, we

extracted information such as daily orders delivered, plant assignments to orders, and the number of available drivers. We created 35 instances from this data, where each instance corresponds to a daily operation. To maintain confidentiality, we have removed the GPS coordinates of customers and plants but have included two matrices containing the distances and driving times between all plants and customers. An instance contains information such as each driver's capacity, associated batch plant, and shift start time (which we generate). For each customer, we have the due time, the total demand, the number of orders (type of concrete) received, and its index in the time (distance) matrix. For each order, we have its demand and the corresponding customer and production plant. Finally, we have the loading capacity for each plant, as well as its index in the time matrix. The name of an instance has the format $C\_n\_o\_k\_p$, where n is the number of customers, o is the total number of orders, k is the number of available drivers, and p is the number of plants. The dataset is divided into small, medium, and large sets according to the total number of daily orders. We have summarized the dataset in Table 2. The small dataset has eight instances with four to eleven customers whose demands vary between 226.5 and 937.5 m^3. These customers request between 7 and 14 orders, which are served from one to three depots with a fleet of 13 to 35 concrete mixers. The dataset is available online at `https://sites.google.com/view/jfcote/` as a contribution to the CDP literature.

Table 2: Instances summary

	#	Demand (m^3)	#Orders	#Customers	#Drivers	#Depots
Small	8	226.5 – 937.5	7 – 14	4 – 11	13 – 35	1 – 3
Medium	14	1,160.5 – 2,971.0	43 – 98	40 – 89	76 – 137	6 – 8
Large	13	3,078.5 – 3,953.5	107 – 136	92 – 114	129 – 149	8

5.2. Results for the instances of Kinable et al. (2014)

Kinable et al. (2014) provided a benchmark dataset for their variant of the CDP. Although their primary objective is to maximize the total load delivered each day without considering loading operations at a plant and ensuring deliveries within specified time windows, we still use this dataset to analyze and compare the performance of our algorithm under different conditions. Kinable et al. (2014) solved their problem with a CP algorithm, MIP, steepest descent heuristic

(SD), and fix-and-optimize MIP heuristic (MIP fix & opt), which is a hybridization of the MIP and SD. They also provide dual bounds to help assess the optimality of the instances. Table 3 shows the performance of the GRASP heuristic against the methods mentioned above. For each method, the table shows the average execution time in seconds and the corresponding gap between the obtained solution and the dual bound provided by Kinable et al. (2014). The row Opt reports the number of optimal solutions found by each method.

Table 3: Performance of the GRASP heuristic on the CDP benchmark instances

		Kinable et al. (2014)								Our paper	
		CP		MIP		SD-heuristic		MIP fix & opt		GRASP	
		Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)
Set A	Avg	4.2	196.95	7.3	149.14	9.1	0.02	7.0	100.70	5.7	13.18
	Opt	40/64		37/64		30/64		35/64		35/64	
Set B	Avg	12.1	355.95	-	-	16.3	1.33	-	-	13.8	574.98
	Opt	55/128		-		40/128		-		44/128	

Overall, our GRASP algorithm shows its effectiveness in providing good and competitive solutions in a relatively short time for this benchmark of the CDP, making it a suitable candidate for further analysis and real-world application. It outperformed all methods except CP, achieving an average gap of 5.7% (13.8%) with a runtime of 13 (575) seconds for Set A (Set B). The SD-heuristic appears to be much faster (23 milliseconds for set A), but GRASP provides a better balance between solution quality and computation time. It should also be noted that the exact MIP, CP, and MIP fix & opt are initialized with the results of the SD-heuristic.

5.3. Results for the generated instances

The computational results of our instances solved with the GRASP heuristic are summarized in the tables below. The parameters used in our experiments are listed in Table 4.

Table 5 contains the detailed results for all instances with a stop criterion of 3600 seconds. The table shows the demand of each instance, the average, and the best values of the objective function when solving each instance five times. All daily orders of the small instances are completely served by our algorithm within these five iterations, as shown in column UQ, which reports the undelivered quantities. For the medium and large instances, 99.5% and 99.6% of the total demands are delivered. The sum of driver underutilization costs (DUC) is high for

Table 4: Parameters of the real-world instances

Parameter	Value
Time window (τ^w)	60 min
Unloading duration (τ_c^u)	2 min/m^3
Cleaning duration (ρ)	10 min
Adjustment duration (α)	10 min
Max travel time (Δ)	120 min
Min working time (μ)	180 min
Max working time (η)	480 min
Max overtime duration (Λ)	120 min
Max delay between two consecutive deliveries of the same order γ^1)	20 min
Max delay between two consecutive deliveries of different orders (γ^2)	25 min
Penalty of unfulfilled orders (β_1)	100,000
Penalty of first delivery delay (β_2)	10,000
Penalty for driver underutilization (β_3)	30
Driver overtime penalty (β_4)	20

medium and large instances. This indicates the underutilization of the scheduled fleet. Some drivers are not scheduled at all or work very little. The sum of driver overtime costs (DOC) is also high for these instances. This can be explained by the fact that the algorithm prioritizes drivers on overtime rather than using a driver who has to travel to another plant for a delivery, which would increase travel costs (TC).

FDD represents the sum of the delays of the first deliveries. In the case of small instances, all but one of the first deliveries are on time. However, for the other instance sets, there is at least one customer with a late first delivery. To provide insight into the actual delay experienced by customers, we report $mFDD$, which is the maximum delay among all first deliveries. This information reveals that for small instances, the maximum delay ranges between 0 and 35.7 minutes. For medium instances, it varies from 32.7 to 60 minutes, and for large instances, it falls within the range of 56 to 60 minutes.

Table 6 shows that a higher percentage of deliveries are completed as the algorithm runtime increases. For small instances, all requests are scheduled in less than a minute of runtime, while for medium and large instances, 99% of requests are scheduled to be delivered with only a minute of runtime.

The medium instance $C\_42\_47\_104\_8$ has an unserved customer with a demand of 12 m^3 and a due time of 07:00. Upon investigation, we find that even though we have 104 drivers

Table 5: Results with all instances

						Avg						Best						
	$\|C\|$	$\|O\|$	$\|K\|$	$\|B\|$	Demand (m^3)	TC (min)	UQ (m^3)	FDD (min)	DUC (min)	DOC (min)	Z	(TC) (min)	UQ (m^3)	FDD (min)	$mFDD$ (min)	DUC (min)	DOC (min)	Z
Small	11	12	13	1	226.5	1,243.5	0.0	0.0	18.8	0.0	1,807.7	1,243.5	0.0	0.0	0.0	0.0	0.0	1,243.5
	5	8	13	1	267.0	745.1	0.0	0.0	0.0	0.0	745.1	745.1	0.0	0.0	0.0	0.0	0.0	745.1
	6	11	18	2	333.5	753.3	0.0	0.0	255.9	0.0	8,430.6	753.3	0.0	0.0	0.0	176.0	0.0	6,034.2
	4	7	15	2	375.0	1,017.5	0.0	0.0	218.2	131.3	10,190.9	1,017.6	0.0	0.0	0.0	82.8	33.9	4,179.8
	7	8	19	2	388.0	1,790.5	0.0	0.0	318.4	137.8	14,119.2	1,817.0	0.0	0.0	0.0	52.6	98.7	5,368.6
	10	14	29	3	613.5	3,482.1	0.0	59.5	239.4	373.5	77,635.1	3,513.2	0.0	59.5	35.7	147.2	304.0	73,510.2
	8	11	31	3	776.0	2,325.7	0.0	0.0	796.3	690.1	40,015.9	2,311.7	0.0	0.0	0.0	593.7	601.0	32,141.2
	9	10	35	3	937.5	4,110.9	0.0	0.0	447.7	712.0	31,782.1	4,139.9	0.0	0.0	0.0	243.3	550.0	22,439.9
			Average		-	1,933.6	0.0	7.4	286.8	255.6	23,090.8	1942.7	0.0	7.4	4.5	161.9	198.5	18,207.8
Medium	40	43	76	6	1,160.5	6,085.6	0.0	44.8	3,017.6	1,140.3	164,186.8	6,451.5	0.0	32.7	8.5	2,714.0	899.1	138,530.5
	42	47	104	8	1,565.0	8,024.3	12.0	55.5	3,563.0	160.2	1,373,593.7	8,106.6	12.0	54.7	16.8	3,358.8	138.7	1,366,363.9
	63	70	94	7	1,746.5	13,871.4	0.0	228.2	1,250.1	1,398.4	307,504.3	13,902.5	0.0	195.3	58.9	1,465.1	1,206.7	277,317.6
	67	78	116	8	1,839.5	13,225.9	0.0	142.1	1,931.4	751.0	228,250.5	13,460.9	0.0	139.3	33.8	935.5	962.9	200,057.2
	71	82	116	8	2,060.0	11,158.1	9.0	263.3	2,797.8	982.1	1,277,996.2	10,748.5	9.0	243.3	45.9	2,218.9	594.8	1,232,488.4
	57	70	117	6	2,327.5	14,551.7	0.0	322.2	1,654.3	1,566.4	417,729.4	14,732.0	0.0	252.5	49.5	1,438.4	1,551.1	341,397.0
	79	83	137	7	2,425.0	17,886.5	0.0	663.2	2,132.2	2,031.5	785,718.4	17,555.1	0.0	559.6	58.6	1,707.6	2,159.2	671,538.1
	66	80	127	8	2,512.5	15,108.4	0.0	118.4	1,600.3	1,596.7	213,496.4	14,567.4	0.0	114.1	34.3	1,349.9	1,143.2	191,985.5
	68	74	128	7	2,595.0	18,744.1	1.6	547.5	1,238.4	2,827.7	822,423.5	18,276.6	0.0	355.6	55.8	1,206.4	3,061.4	471,318.0
	85	97	136	8	2,673.0	17,252.5	0.0	427.7	2,085.1	2,355.1	554,640.8	18,164.5	0.0	361.0	53.3	1,447.8	2,081.4	464,253.2
	78	85	128	8	2,685.5	15,931.9	63.0	461.4	1,243.9	2,783.1	6,870,259.9	16,164.5	63.0	391.1	57.8	1,553.3	2,371.0	6,801,308.3
	77	85	131	8	2,893.5	17,168.9	0.0	410.7	1,360.1	1,756.7	503,850.9	17,620.2	0.0	388.1	57.5	963.2	2,149.1	477,607.4
	89	97	137	7	2,939.5	21,350.0	13.4	865.0	1,074.2	3,247.2	2,327,223.6	21,551.0	0.0	880.5	47.7	1,318.9	3,375.7	1,009,098.9
	84	98	133	7	2,971.0	20,620.2	39.2	1,091.8	736.8	2,666.4	5,106,602.3	20,578.3	31.5	887.0	59.1	839.1	2,910.7	4,141,019.3
			Average		-	15,070.0	9.9	403.0	1,834.7	1,804.5	1,496,676.9	15,134.3	8.2	346.8	45.5	1,608.4	1,757.5	1,270,305.9
Large	98	109	132	8	3,078.5	22,498.9	0.0	456.2	457.4	3,961.9	571,626.0	22,359.3	0.0	394.4	56.2	463.1	3,905.6	508,725.2
	101	119	129	8	3,229.0	17,814.1	9.6	647.3	1,342.6	2,674.1	1,721,369.9	18,811.0	0.0	608.8	57.5	969.3	2,715.1	710,970.8
	114	136	141	8	3,350.5	20,596.8	44.7	1,346.5	1,048.3	2,672.2	5,920,713.6	20,811.8	33.5	1,453.0	56.6	616.4	1,788.8	4,878,101.5
	114	132	140	8	3,401.5	22,016.2	27.7	1,320.5	810.1	3,021.1	4,196,040.1	22,693.0	14.5	1,487.4	58.6	525.9	2,616.0	3,028,191.3
	101	123	143	8	3,437.5	23,804.3	0.0	649.5	942.7	4,558.3	792,752.6	22,514.1	0.0	447.0	59.1	1,561.5	3,605.0	588,441.4
	112	129	137	8	3,471.0	21,873.0	0.0	520.3	613.1	3,935.4	639,262.4	21,352.6	0.0	473.6	52.4	958.4	3,604.6	595,823.5
	114	129	142	8	3,499.5	21,173.3	28.1	1,119.6	994.0	4,116.9	4,065,444.5	21,178.8	8.0	1,042.9	58.1	757.2	4,141.9	1,969,653.9
	98	122	149	8	3,513.0	19,739.2	35.7	1,141.4	1,842.7	2,570.9	4,836,612.0	19,413.8	18.5	1,233.5	59.2	2,192.8	2,399.4	3,216,666.5
	98	108	139	8	3,541.0	20,862.3	0.0	941.6	628.4	3,526.0	1,051,844.1	20,241.0	0.0	847.2	59.1	1,106.1	2,790.2	956,432.6
	108	122	144	8	3,670.5	22,605.8	2.1	1,127.8	877.3	2,790.1	1,445,018.0	22,369.7	0.0	873.3	56.0	1,126.8	2,922.1	987,898.0
	92	107	142	8	3,684.5	22,877.9	0.0	584.4	638.0	4,754.3	721,474.4	23,511.5	0.0	458.5	53.4	593.9	4,715.3	594,093.8
	114	136	138	8	3,739.5	23,361.3	8.5	1,178.1	853.7	5,271.4	2,182,537.5	23,633.6	0.0	961.3	56.1	766.6	5,434.9	1,116,585.2
	112	131	148	8	3,953.5	24,313.5	34.9	1,251.9	524.6	2,961.1	4,838,688.0	25,301.2	13.5	1,694.5	59.4	766.0	2,944.6	3,151,661.8
			Average		-	21,810.5	14.7	945.0	890.2	3,601.0	2,537,183.3	21,860.9	6.8	921.2	57.1	954.2	3,352.6	1,715,634.3

TC: travel cost, UQ: undelivered quantity, FDD: first delivery delay, $mFDD$: maximum of first delivery delays, DUC: driver underutilization cost, DOC: driver overtime cost.

Table 6: RMC delivery completion within different runtimes of the GRASP

		1 min	5 min	10 min	30 min	60 min
Small	%Load	100.0	100.0	100.0	100.0	100.0
Medium	%Load	98.9	99.3	99.4	99.5	99.5
Large	%Load	98.8	99.3	99.4	99.6	99.6

in this instance, the number of drivers who start their shift before 07:00 is insufficient to serve this customer within the 60-minute window we defined. Thus, the value of UQ that we report in Table 5 appears optimal given the driver shift schedule provided as input to our algorithm. We confirmed this observation by removing the driver start shift and overtime constraints and resolving our instances five more times. All orders are now fully serviced on each run, as shown in Table 7. Table 8 compares the averages of the results from Tables 5 and 7. We found that

the maximum delay dropped to zero for small instances, and for medium (large) instances, it was reduced by almost 60% (40%) on average. This suggests that when we input a good work schedule, our algorithm works effectively to ensure that all orders are filled and that delays for first deliveries and travel costs are minimized. We can see that the average travel time decreases in all cases except the large one, where the difference is not significant.

Table 7: Results with all instances without shift and overtime constraints

						Avg				Best				
	$\|C\|$	$\|O\|$	$\|K\|$	$\|B\|$	Demand (m^3)	TC (min)	UQ (m^3)	FDD (min)	Z	TC (min)	UQ (m^3)	FDD (min)	$mFDD$ (min)	Z
Small	11	12	13	1	226.5	1,243.5	0.0	0.0	1,243.5	1,243.5	0.0	0.0	0.0	1,243.5
	5	8	13	1	267.0	745.1	0.0	0.0	745.1	745.1	0.0	0.0	0.0	745.1
	6	11	18	2	333.5	753.3	0.0	0.0	753.3	753.3	0.0	0.0	0.0	753.3
	4	7	15	2	375.0	1,017.6	0.0	0.0	1,017.6	1,016.5	0.0	0.0	0.0	1,016.5
	7	8	19	2	388.0	1,662.7	0.0	0.0	1,662.7	1,651.8	0.0	0.0	0.0	1,651.8
	10	14	29	3	613.5	3,259.0	0.0	0.0	3,259.0	3,200.9	0.0	0.0	0.0	3,200.9
	8	11	31	3	776.0	2,222.0	0.0	0.0	2,222.0	2,208.9	0.0	0.0	0.0	2,208.9
	9	10	35	3	937.5	3,839.6	0.0	0.0	3,839.6	3,750.5	0.0	0.0	0.0	3,750.5
			Average		-	1,822.7	0.0	0.0	1,822.7	1,821.3	0.0	0.0	0.0	1,821.3
Medium	40	43	76	6	1,160.5	5,242.1	0.0	0.0	5,242.1	5,213.5	0.0	0.0	0.0	5,213.5
	42	47	104	8	1,565.0	8,291.5	0.0	0.0	8,291.5	7,993.7	0.0	0.0	0.0	7,993.7
	63	70	94	7	1,746.5	12,744.2	0.0	0.0	12,744.2	12,638.2	0.0	0.0	0.0	12,638.2
	67	78	116	8	1,839.5	11,918.0	0.0	0.0	11,919.8	11,793.3	0.0	0.0	0.0	11,793.3
	71	82	116	8	2,060.0	10,612.5	0.0	0.0	10,612.5	10,449.9	0.0	0.0	0.0	10,449.9
	57	70	117	6	2,327.5	14,781.6	0.0	15.7	30,457.6	14,391.0	0.0	2.4	1.4	16,820.6
	79	83	137	7	2,425.0	17,637.5	0.0	106.5	124,143.7	17,504.0	0.0	66.7	21.7	84,156.1
	66	80	127	8	2,512.5	13,676.8	0.0	0.0	13,676.8	13,595.0	0.0	0.0	0.0	13,595.0
	68	74	128	7	2,595.0	18,565.9	0.0	97.4	115,988.9	18,484.6	0.0	72.3	29.0	90,740.3
	85	97	136	8	2,673.0	15,741.9	0.0	75.7	91,395.7	15,779.3	0.0	48.2	18.6	64,002.6
	78	85	128	8	2,685.5	17,097.4	0.0	68.0	85,088.9	16,326.4	0.0	55.6	14.0	71,906.4
	77	85	131	8	2,893.5	17,807.8	0.0	10.7	28,544.5	17,994.5	0.0	9.4	6.1	27,404.5
	89	97	137	7	2,939.5	20,810.3	0.0	203.9	224,680.9	20,829.3	0.0	150.0	42.0	170,861.3
	84	98	133	7	2,971.0	21,142.9	0.0	380.8	401,955.9	21,206.1	0.0	252.7	55.6	273,892.1
			Average		-	14,719.3	0.0	68.5	83,195.9	14,585.6	0.0	46.9	13.5	61,533.4
Large	98	109	132	8	3,078.5	22,186.9	0.0	15.2	37,379.4	22,061.8	0.0	9.9	6.3	31,964.6
	101	119	129	8	3,229.0	18,290.6	0.0	40.5	58,817.5	17,111.2	0.0	10.5	5.7	27,595.2
	114	136	141	8	3,350.5	21,244.6	0.0	122.4	143,618.2	21,702.8	0.0	90.0	58.9	111,699.9
		132	140	8	3,401.5	22,888.6	0.0	119.9	142,811.7	23,079.3	0.0	78.7	16.9	101,807.4
	101	123	143	8	3,437.5	24,070.1	0.0	44.1	68,139.4	23,863.3	0.0	13.7	11.9	37,577.6
	112	129	137	8	3,471.0	22,497.5	0.0	120.4	142,915.9	21,867.0	0.0	104.6	16.4	126,500.0
	114	129	142	8	3,499.5	21,545.5	0.0	461.2	482,716.9	20,476.5	0.0	236.3	53.1	256,802.5
	98	122	149	8	3,513.0	20,370.9	0.0	150.5	170,865.5	19,725.6	0.0	106.5	46.3	126,172.6
		108	139	8	3,541.0	21,538.2	0.0	339.1	360,638.8	21,205.7	0.0	201.3	59.5	222,493.7
	108	122	144	8	3,670.5	23,462.8	0.0	253.3	276,751.0	23,827.4	0.0	144.9	31.9	168,731.4
	92	107	142	8	3,684.5	22,799.8	0.0	52.4	75,149.3	24,071.4	0.0	30.9	10.2	54,975.8
	114	136	138	8	3,739.5	23,491.1	0.0	86.7	110,190.5	23,242.2	0.0	46.7	32.5	69,982.2
	112	131	148	8	3,953.5	25,300.6	0.0	387.0	412,296.4	26,296.6	0.0	247.8	32.5	274,119.6
			Average		-	22,454.1	0.0	187.5	209,966.8	22,504.4	0.0	123.6	26.8	146,138.2

TC: travel cost, UQ: undelivered quantity, FDD: first delivery delay, $mFDD$: maximum of first delivery delays.

Table 8: Comparison of the average results with and without shift and overtime constraints

	Default parameters				$\mu = 0$, $\eta, \Lambda = \infty$, $h_k = 0$			
	TC (min)	UQ (m^3)	FDD (min)	$mFDD$ (min)	(TC) (min)	UQ (m^3)	FDD (min)	$mFDD$ (min)
Small	1,933.6	0.0	7.4	4.5	1,822.7	0.0	0.0	0.0
Medium	15,070.0	9.9	403.0	46.9	14,719.3	0.0	68.5	18.4
Large	21,810.5	14.7	945.0	56.4	22,283.6	0.0	168.7	31.0

TC: travel cost, UQ: undelivered quantity, FDD: first delivery delay, $mFDD$: maximum of first delivery delays.

5.4. Sensitivity analyses

In this section, we analyze the effect on the solution when we change some parameters of our problem and how some objective function components influence each other.

5.4.1. Influence of the maximum time delay γ^1

We ran additional tests with different values of γ^1, varying between 5 and 25 minutes, with no shifts or overtime constraints on the drivers. The influence of the value of the maximum time delay between two consecutive deliveries of the same order is shown in Figure 2. For small instances, all customers are served on time regardless of the value of γ^1. For medium (large) instances, all customers are served with γ^1 starting at 15 (20) minutes. The delay of initial delivery decreases as γ^1 increases. These results confirm the fact that when the maximum time delay γ^1 is high, plants have more flexibility to delay current deliveries to make room for new ones or to make customers wait until drivers are available. Increasing the maximum delay from 10 to 25 minutes reduces the first delivery delay by over 50% for the medium and large cases. The average delivery time does not change significantly regardless of the value of γ^1.

5.4.2. Trade-off of the objective function components

We ran five tests on our dataset and obtained two solutions per instance: one that minimizes the travel time and another that minimizes the first delivery delay. Table 9 reports the trade-off between the travel time and the first delivery delay for these solutions. The difference is not noticeable for small instances. However, for medium and large instances, it is clear that as the first delivery delay decreases, the travel time increases, and conversely, when we minimize the travel time, the first delivery delay tends to be longer. This result is expected because drivers

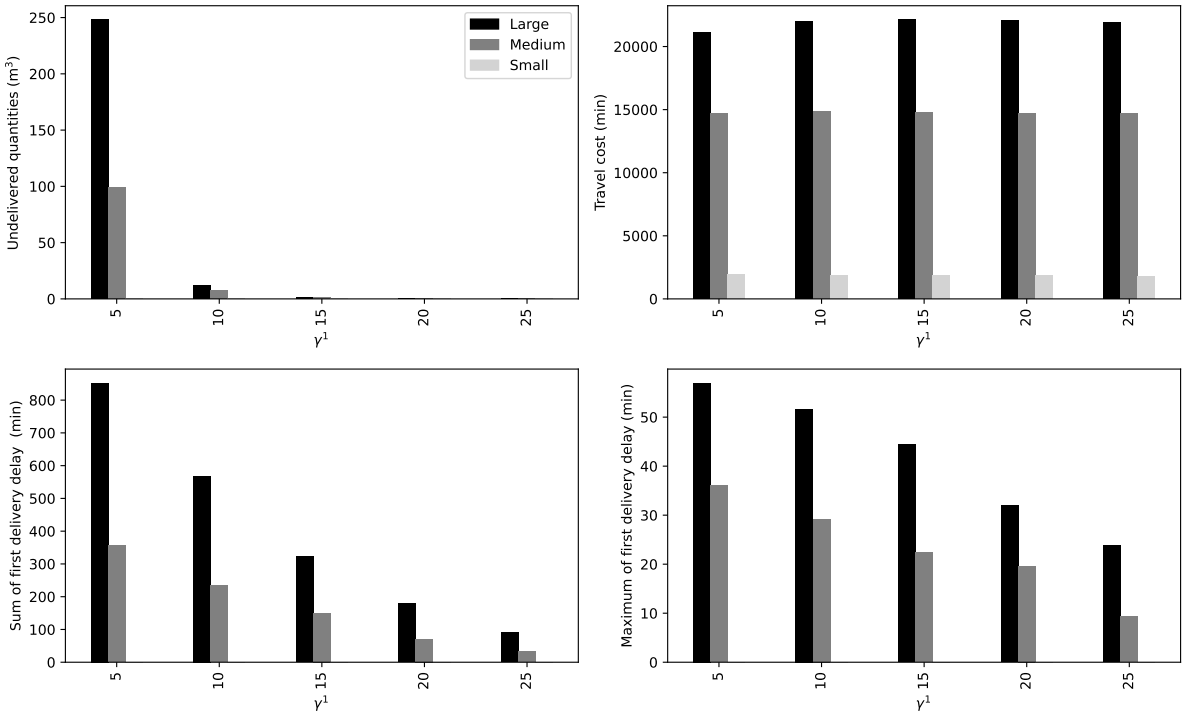


Figure 2: Influence of the maximum time delay (γ^1) between consecutive deliveries of the same order.

assigned to one plant often must travel to another one to ensure that the first delivery remains on schedule. This trade-off highlights the challenge decision-makers face in finding the right balance between minimizing operational costs and meeting customer satisfaction with high service levels.

Table 9: Trade-off between travel cost and first delivery delay

	Minimize TC			Minimize FDD		
	TC (min)	FDD (min)	$mFDD$ (min)	TC (min)	FDD (min)	$mFDD$ (min)
Small	1,821.3	0.0	0.0	1,831.6	0.0	0.0
Medium	14,311.5	89.6	23.0	14,679.3	46.9	13.4
Large	21,403.8	170.5	31.4	22,194.7	101.7	29.4

6. Conclusion

In this paper, we investigated a variant of the concrete delivery problem, where customers can order different types of concrete from different production centers, which must be delivered

during the same time window. To solve this problem, we introduced a mathematical model and a heuristic solution based on Greedy Randomized Adaptive Search. Our analysis, using a dataset derived from our partner's historical data, demonstrated the effectiveness of our algorithm in providing good solutions within a reasonable time. Furthermore, our model incorporates constraints related to working hours, making our heuristic a valuable tool for designing weekly schedules for company drivers. It can also better reassign drivers to their home plants, as these assignments affect travel costs to ensure timely deliveries. Our GRASP algorithm performed exceptionally well on the Kinable et al. (2014) datasets, indicating its reliability. For future work, it would be interesting to address datasets with different unloading times at construction sites or to explore different scenarios regarding the number of simultaneous deliveries and the maximum time between two consecutive orders at a construction site. One can also study the problem using a stochastic or robust methodology to account for the uncertainty of the parameters in a real-world setting. As a contribution to the literature on CDP, we provide a mathematical formulation for our variant, contribute with a dataset, and demonstrate the application of GRASP to solve both our real problem and the one studied by Kinable et al. (2014).

Acknowledgments

Financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC) under grants 2019-00094 and 2021-04037. This support is gratefully acknowledged. We thank the Digital Research Alliance of Canada for providing high-performance computing facilities.

References

A. Sinha, N. Singh, G. Kumar, S. Pal, Quality factors prioritization of ready-mix concrete and site-mix concrete: A case study in indian context, in: D. Singh, A. K. Awasthi, I. Zelinka, K. Deep (Eds.), Proceedings of International Conference on Scientific and Natural Computing, Algorithms for Intelligent Systems, Springer, Singapore, 2021, pp. 179–187.

F. Muresan, Comparing ready-mix concrete and site-mixed concrete, 2019. URL: https://www.ny-engineers.com/blog/ready-mix-concrete-and-site-mixed-concrete.

M. Durbin, K. Hoffman, Or practice—the dance of the thirty-ton trucks: Dispatching and scheduling in a dynamic environment, Operations Research 56 (2008) 3–19.

V. Schmid, K. F. Doerner, R. F. Hartl, M. W. P. Savelsbergh, W. Stoecher, A hybrid solution approach for ready-mixed concrete delivery, Transportation Science 43 (2009) 70–85.

V. Schmid, K. F. Doerner, R. F. Hartl, J.-J. Salazar-González, Hybridization of very large neighborhood search for ready-mixed concrete delivery problems, Computers & Operations Research 37 (2010) 559–574.

J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, M. Sterna, J. Weglarz, Handbook on scheduling: From theory to practice, Springer, 2019.

C. Archetti, M. G. Speranza, The split delivery vehicle routing problem: a survey, in: The vehicle routing problem: Latest advances and new challenges, Springer, 2008, pp. 103–122.

I. Tommelein, A. Li, Just-in-time concrete delivery: mapping alternatives for vertical supply chain integration, in: Proceedings of the 7th Annual Conference of the International Group for Lean Construction, volume 7, Berkeley, University of California, 1999, pp. 97–108.

T. M. Zayed, D. Halpin, Simulation of concrete batch plant production, Journal of Construction Engineering and Management 127 (2001) 132–141.

C.-W. Feng, T.-M. Cheng, H.-T. Wu, Optimizing the schedule of dispatching rmc trucks through genetic algorithms, Automation in Construction 13 (2004) 327–340.

M. Lu, H.-C. Lam, Optimized concrete delivery scheduling using combined simulation and genetic algorithms, in: Proceedings of the Winter Simulation Conference, IEEE, 2005.

M. Maghrebi, S. T. Waller, C. Sammut, Sequential meta-heuristic approach for solving large-scale ready-mixed concrete–dispatching problems, Journal of Computing in Civil Engineering 30 (2016) 04014117.

J. Yang, B. Yue, F. Feng, J. Shi, H. Zong, J. Ma, L. Shangguan, S. Li, Concrete vehicle scheduling based on immune genetic algorithm, Mathematical Problems in Engineering (2022).

L. Asbach, U. Dorndorf, E. Pesch, Analysis, modeling and solution of the concrete delivery problem, European Journal of Operational Research 193 (2009) 820–835.

J. Kinable, T. Wauters, G. V. Berghe, The concrete delivery problem, Computers & Operations Research 48 (2014) 53–68.

M. Maghrebi, V. Periaraj, S. T. Waller, C. Sammut, Using Benders decomposition for solving ready mixed concrete dispatching problems, in: The 31st International Symposium on Automation and Robotics in Construction and Mining, 2014a.

M. Maghrebi, V. Periaraj, S. T. Waller, C. Sammut, Solving ready-mixed concrete delivery problems: Evolutionary comparison between column generation and robust genetic algorithm, in: Computing in Civil and Building Engineering (2014), 2014b, pp. 1417–1424.

P. K. Narayanan, D. Rey, M. Maghrebi, S. T. Waller, Using Lagrangian relaxation to solve ready mixed concrete dispatching problems, Transportation Research Record 2498 (2015) 84–90.

L. D. Graham, D. R. Forbes, S. D. Smith, Modeling the ready mixed concrete delivery system with neural networks, Automation in Construction 15 (2006) 656–663.

N. F. Matsatsinis, Towards a decision support system for the ready concrete distribution system: A case of a greek company, European Journal of Operational Research 152 (2004) 487–499.

D. Naso, M. Surico, B. Turchiano, U. Kaymak, Genetic algorithms for supply-chain scheduling: A case study in the distribution of ready-mixed concrete, European Journal of Operational Research 177 (2007) 2069–2099.

S. Yan, W. Lai, An optimal scheduling model for ready mixed concrete supply with overtime considerations, Automation in Construction 16 (2007) 734–744.

J. Garza Cavazos, Dynamic planning and real-time monitoring of ready-mixed concrete delivery problem, Ph.D. thesis, Universidad Autónoma de Nuevo León, 2021.

F. Payr, V. Schmid, Optimizing deliveries of ready-mixed concrete, in: 2009 2nd International Symposium on Logistics and Industrial Informatics, 2009, pp. 1–6.

A. Tzanetos, M. Blondin, Systematic search and mapping review of the concrete delivery problem (CDP): Formulations, objectives, and data, Automation in Construction 145 (2023) 104631.

T. A. Feo, M. G. Resende, A probabilistic heuristic for a computationally difficult set covering problem, Operations Research Letters 8 (1989) 67–71.

M. Bamoumen, S. Elfirdoussi, L. Ren, N. Tchernev, An efficient GRASP-like algorithm for the multi-product straight pipeline scheduling problem, Computers & Operations Research 150 (2023) 106082.

J. C. Yepes-Borrero, F. Perea, F. Villa, E. Vallada, Flowshop with additional resources during setups: Mathematical models and a GRASP algorithm, Computers & Operations Research 154 (2023) 106192.

D. Ferone, P. Festa, T. Pastore, M. G. Resende, Efficient GRASP solution approach for the prisoner transportation problem, Computers & Operations Research 153 (2023) 106161.

J. G. Villegas, C. Prins, C. Prodhon, A. L. Medaglia, N. Velasco, A GRASP with evolutionary path relinking for the truck and trailer routing problem, Computers & Operations Research 38 (2011) 1319–1334.