A Hybrid Solution Approach for Ready-Mixed Concrete Delivery

Verena Schmid* Karl F. Doerner* Richard F. Hartl*

Martin W.P. Savelsbergh† Wolfgang Stoecher‡

*Department of Business Administration, University of Vienna

Bruenner Strasse 72, A-1210 Wien

verena.schmid,karl.doerner,richard.hartl@univie.ac.at

†H. Milton Stewart School of Industrial and Systems Engineering,

Georgia Institute of Technology

Atlanta, GA 30332-0205

mwps@isye.gatech.edu

‡Profactor Produktionsforschungs GmbH

Im Stadtgut A2, A-4407 Steyr-Gleink

wolfgang.stoecher@profactor.at

Abstract

Companies in the concrete industry are facing the following scheduling problem on a daily basis: concrete produced at several plants has to be delivered at customers' construction sites using a heterogeneous fleet of vehicles in a timely, but cost-effective manner. As the ordered quantity of concrete typically exceeds the capacity of a single vehicle several deliveries need

to be scheduled in order to fulfill an order. The deliveries cannot overlap and the time between consecutive deliveries has to be small. Our solution approach effectively integrates optimization and heuristic techniques. Information is passed back and forth between an integer multi-commodity flow optimization component and a variable neighborhood search component in order to find high-quality solutions in a reasonable amount of time. Even though both components are capable of producing feasible solutions, the integrated approach is far more effective. Computational results show that our hybrid approach outperforms an innovative metaheuristic approach by more than 6 percent on average for large instances.

Keywords: hybrid approach, variable neighborhood search, integer multi-commodity flow, ready-mixed concrete delivery

1 Introduction and Problem Setting

Concrete is needed almost everywhere. In order to build factories, commercial and residential buildings, etc., some type of construction material is needed, and concrete is one of the preferred and most used construction materials (see ERMCO [19]).

The ready-mixed concrete market is a prosperous one. The total amount of concrete produced in the European Union has increased from 318.4 million m^3 in 2002 to 369.6 million m^3 in 2005, an increase of 16.08%. Similar quantities are produced in the United States, where the total amount of concrete produced has risen by 15% up to 345 million m^3 per year in the same period (see ERMCO [20, 21]). Emerging markets, such as China and India, push up the demand for concrete even more.

Concrete is produced by blending cement, aggregates, such as gravel and sand, and water. Additionally, certain admixtures, e.g. retarders and accelerators, are added in order to affect the hydration or hardening process of the material. Depending on the purpose of the construction being built, other ingredients may be used or added in order to improve water permeability, to change the color, etc. Concrete is a perishable good, in a sense that it hardens after a certain

amount of time. During the blending process it is smooth and it can be transported for some time if it remains in movement. After about two hours, depending on accelerators or retarders in use, concrete hardens and it will obtain its required durability and strength.

As the name suggests, ready-mixed concrete is not produced at the construction sites where it is needed. Production takes place at plants from where the ready-mixed concrete is transported to the construction sites using vehicles specifically designed to transport concrete. The concrete is mixed just-in-time before the loading of the vehicle at the plant, or the raw materials are poured into the vehicle and are mixed on the way to the construction site. Concrete is a perishable good which cannot be stored or produced in advance. Moreover after being blended the good cannot be transported for more than two hours.

Companies in the concrete industry are facing the following scheduling problem on a daily basis: concrete produced at several plants has to be delivered at customers' construction sites using a heterogeneous fleet of vehicles in a timely, but cost-effective manner. As the ordered quantity of concrete typically exceeds the capacity of a single vehicle, several deliveries need to be scheduled in order to fulfill an order. The delivery process is complicated by the fact that at most one vehicle can unload at a construction site at a time, i.e., deliveries cannot overlap, and that the time between consecutive deliveries cannot be too long. Some orders require vehicles with special equipment to be present for the delivery of concrete, e.g., a pump or a conveyer belt. Such vehicles need to arrive first at a construction site and remain at the construction site until the complete order has been fulfilled. Some of the vehicles with special equipment may also be able to carry concrete. As a result, a mix of vehicles is available and has to be properly managed. The typical objective is to fulfill all orders, to minimize the travel cost, and to avoid delays between two consecutive unloading operations for an order.

The variant of ready-mixed concrete delivery we study is motivated by the operations of a medium size concrete company located in Alto Adige, Italy. In their environment, all orders to be satisfied on a particular day are known the evening before and the delivery schedule is

determined during the night. When placing an order, customers specify a preference for the start time of the first delivery. To ensure customer satisfaction, a time window is assigned to every order indicating the period during which the first delivery should start. The fleet of vehicles at the company is large enough to satisfy all orders in a timely manner. The time needed for loading and unloading a vehicle depends on the vehicle's capacity and a plant's loading and a construction site's unloading rate, respectively. Each vehicle serves one order at a time. The fleet of vehicles is heterogenous. First, the vehicles may differ in terms of capacity. Second, the vehicles may differ in terms of their instrumentation. Some vehicles can only be used for the delivery of concrete. Others might only be used to assist during unloading operations, e.g., by providing a pump or a conveyor belt. Finally, there are hybrid vehicles, which are used for delivery of concrete and for providing the required equipment during unloading operations.

It should come as no surprise that constructing low-cost ready-mixed concrete delivery schedules is extremely challenging. We discuss a novel approach for creating such schedules by integrating integer multi-commodity network flow (MCNF) techniques and variable neighborhood search (VNS) techniques. Our computational experiments demonstrate that the hybridization of these two well-known optimization techniques pays off. Delivery schedules can be obtained by employing either the integer MCNF component or the VNS component, but the resulting schedules are not nearly as good as those that can be obtained by the integrated solver in the same amount of time. Furthermore, our hybrid method outperforms a commercially developed tailored solution approach based on simulated annealing by more than 35 percent on average. The hybridization enables us to combine the strengths of both techniques and compensate for their weaknesses. The VNS is capable of efficiently and effectively exploring the solution space around a known feasible solution. The integer MCNF optimizer takes a global view and is able to jump to promising parts of the solution space.

The integration of optimization techniques and local search techniques has great potential for effectively solving instances of complex problems and is a promising area of research (see

Puchinger and Raidl [23] for a survey of hybrid methods and De Franceschi et al. [4] for an innovative, clever, and successful implementation for the capacitated vehicle routing problem). We believe that our efforts in the context of ready-mixed concrete delivery reenforce that view.

The remainder of the paper is organized as follows. In Section 2, we give an overview of the relevant literature. In Section 3, we discuss the integer MCNF component. In Section 4, we introduce the VNS component. The hybridization, i.e., the integration of the integer MCNF component and the VNS component, is described in Section 5. A comprehensive computational study is presented in Section 6.

2 Literature

Some related work on scheduling and dispatching trucks for the delivery of concrete can be found in the literature. An overview of the main characteristics of the delivery and production of ready-mixed concrete (RMC) can be found in Tommelein and Li [25]. They consider RMC delivery as a prototypical example of a just-in-time production system with batching based on customers demand.

Matsatsinis [14] presents an approach for designing a decision support system for the dynamic routing of trucks in order to distribute ready-mixed concrete. Matsatsinis concentrates on the decision support system; routing is done using heuristics. Different from our approach, Matsatsinis separates the scheduling of vehicles with special delivery equipment and concrete-carrying vehicles. Furthermore, vehicles fulfilling the same order all have to load at the same plant.

Naso et al. [17] implement a two-phase approach. In a preprocessing step orders are split into several jobs (deliveries) based on the vehicle capacity. In the first phase, the jobs and their associated loading operations are assigned to plants using a genetic algorithm. In the second phase, the routing of the vehicles is determined. This is done using a construction heuristic,

which produces a feasible delivery schedule. They assume a homogeneous fleet of vehicles used solely for the delivery of concrete. Specialized unloading equipment, such as pumps or conveyor belts, which may be required during the unloading operation of other vehicles is not considered. Time windows need to be respected and an uninterrupted supply of concrete is required. To handle bottleneck situations that may occur when many tight time windows have to be respected, they allow outsourcing of production and hiring vehicles externally. Their objective has various components: transportation costs, in terms of distance traveled, loading and unloading waiting times, outsourcing costs, and overtime costs. The approach has been extended in Naso et al. [18] to handle additional practical considerations, such as plant capacities and vehicle speeds. A non-linear mathematical model is presented. Furthermore, an event-driven rescheduling approach is proposed to handle perturbations that might occur during the planning horizon.

Durbin [5] and Hoffman and Durbin [10] develop an optimization-based decision-support tool. They present a time-space network formulation and use minimum-cost network flow optimization techniques and tabu search to solve the problem. Their effective use of a time-space network formulation inspired us to include a integer MCNF component in our approach. However, our focus on fulfillment patterns for orders is quite different from their pure network flow formulation.

Asbach et al. [2] develop a general mixed integer programming model, which can only solve very small instances. Hence, a local search based approach is implemented to handle real-world instances. A subset of customers and their associated deliveries are removed from the current solution and then sequentially re-introduced while keeping the timing of all other loading and unloading operations unchanged.

Schmid et al. [24] present a hybrid method combining VNS with integer programming. Integer programming is used to optimally solve small capacitated vehicle routing problems with multiple time windows. The approach produces competitive results for up to medium-sized instances.

Unlike most models proposed in the literature, we consider a heterogenous fleet of vehicles.

Because of the different vehicle capacities, the number of deliveries required to fulfil an order cannot be determined in advance. The additional flexibility significantly complicates constructing feasible low-cost delivery schedules. Furthermore, we consider vehicles with specialized unloading equipment. These vehicles need to arrive first at the construction site so as to assist other vehicles arriving later with their unloading operations. Because there are multi-purpose vehicles, which can deliver concrete and assist during unloading operations, the problem cannot simply be split into two phases (e.g., routing the vehicles with special loading equipment followed by concrete delivery vehicles).

The success of our solution approach is due to the effective integration of two well-known and efficient techniques: IP and VNS. IP is used to solve an integer MCNF formulation, which is instantiated by suggestions produced by the VNS.

Integer MCNF formulations are often used in transportation problems (Ahuja et al. [1], Glover et al. [7]). VNS has proven its effectiveness on a multitude of problems. Efficient implementations for solving routing problems have been developed by Kytöjoki et al. [13]. The vehicle routing problem with time windows has been tackled using VNS by Bräysy [3], the vehicle routing problem with multiple depots and time windows by Polacek et al. [22] and the periodic vehicle routing problem by Hemmelmayr et al. [9].

3 Integer Multi-Commodity Flow Component

The ready-mixed concrete delivery problem can be modeled as an integer MCNF problem on a time-space network (with some similarities to the model proposed by Hoffman and Durbin [10]).

The nodes in the time-space network represent the construction sites at which concrete needs to be delivered at discrete points in time. We have employed a time discretization of five minutes. Links in the network represent possible movements of delivery vehicles. We group vehicles with

the same capacity, the same home plant, and the same instrumentation into vehicle classes and each vehicle class is a commodity.

A crucial entity in the model is a fulfillment pattern for an order. A fulfillment pattern a for order o completely specifies a set of unloading operations that will feasibly fulfill the demand associated with that order, i.e., the exact sequence of vehicles to show up at the construction site as well as the exact points in time when these vehicles will arrive at the construction site and start unloading. Any additional requirements of the order concerning specialized unloading equipment will also be satisfied. The set of fulfillment patterns for order o is denoted by A_o.

If a vehicle of class c starts unloading at time t in fulfillment pattern a for order o, then the binary indicator P_{oa}^{ct} is equal to 1 (and 0 otherwise). For convenience, we denote the start (end) of the first (last) unloading operation in fulfillment pattern a for order o by $start_{oa}$ (end_{oa}). Furthermore, the class of the vehicle that will perform the first unloading operation in fulfillment pattern a for order o is denoted by $first_{oa}$. Every fulfillment pattern is feasible in the sense that the first vehicle does not arrive too early and has any required special equipment, consecutive unloading operations do not overlap, and the cumulative capacity of all vehicles scheduled to make a delivery exceeds the demand for concrete.

A graphical representation of a valid pattern is depicted in Figure 1.

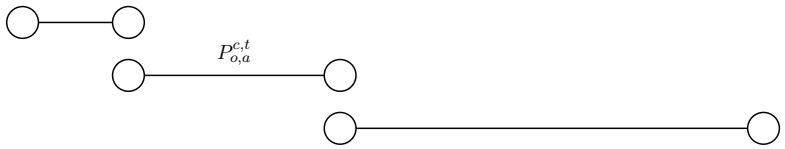

Figure 1: Valid Pattern

Figure 2 also depicts a valid pattern. This time there is a delay between two consecutive unloading operations. Even though undesirable, such patterns are feasible.

In case special unloading equipment is required, the first vehicle to arrive needs to have the proper instrumentation. This vehicle performs its unloading operation and then remains at the

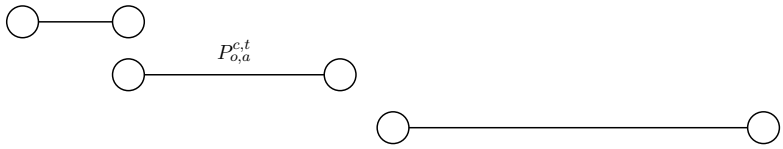

Figure 2: Valid Pattern with Delay

construction site (dotted line) to assist later arriving vehicles with their unloading operation. The vehicle is only allowed to leave after the last vehicle has finished its unloading operation. This situation is depicted in Figure 3.

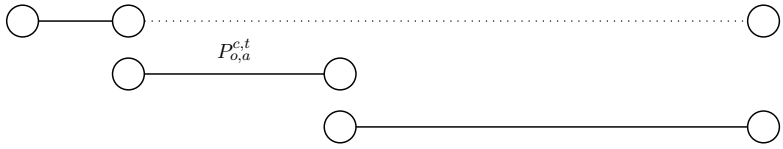

Figure 3: Valid Pattern with Special Instrumentation

The goal is to choose an appropriate fulfillment pattern for each order. Therefore, the following decision variable is introduced

z_{oa}**:** a binary variable indicating whether or not pattern a ($a \in A_o$) is chosen for order o.

and the following set of constraints is introduced

$$\sum_{a \in A_o} z_{oa} \leq 1 \quad \forall o \in O. \tag{1}$$

Vehicle movement between the end of an unloading operation for one order and the start of an unloading operation of another order is modelled as a link in the time-space network. The loading operation and the plant where this loading operation take places are embedded in this link. As the capacity of the plants is assumed not be limiting, it is simple to determine the most cost-effective plant to visit in between two unloading operations.

This results in three types of decision variables (representing flow on the links in the time-space network):

u_o^{ct}: a binary variable indicating whether a vehicle of class c starts its loading operation at its home plant at time t in order to go to the construction site associated with order o for an unloading operation there. (This refers to the first loading operation of the day for the vehicle.)

v_o^{ct}: a general integer variable indicating the number of vehicles of class c ending their unloading operation at the construction site associated with order o at time t in order to go back to their home plant. (This refers to the last unloading operation of the day for the vehicles.)

$x_{o_1o_2}^{ct}$: a binary variable indicating whether a vehicle of class c leaves the construction site associated with order o_1 at t to go to the construction site associated with order o_2 for an unloading operation there. (This refers to the vehicle movements throughout the day, except for the first and the last one.)

Once a fulfillment pattern $a \in A_o$ is chosen for order $o \in O$, we have to ensure that the unloading operations of that pattern are going to be performed, i.e., we have to ensure that the appropriate vehicle arrives at the appropriate time. The model assumes that vehicles arrive just in time, i.e., any waiting happens at the location from which the vehicle departs, either a plant or a construction site. (In practice, of course, vehicles will not remain idle at construction sites after finishing their unloading operations, but will immediately drive to the plant where they will be loaded next. For modeling purposes, though, it is more convenient to assume that waiting happens at the location from which the vehicle departs.) Vehicles will start their unloading operation immediately after arriving at the construction site. Let tt_{po}^c denote the time required for a vehicle of class c to load at its home plant p plus the driving time to the construction site associated with order o. Furthermore, let $tt_{o_1o_2}^c$ denote the driving time from the construction site associated with order o_1 to the construction site associated with order o_2 plus the loading time enroute at the closest plant (within two hours from the construction site of o_2) for a vehicle of class c. The following set of constraints ensures the arrival at the right time of the right

vehicle for an order and its associated pattern:

$$\sum_{a \in A_o} P_{oa}^{ct} z_{oa} = u_o^{c,t-tt_{po}^c} + \sum_{o_1 \in O} x_{o_1 o}^{c,t-tt_{o_1 o}^c} \quad \forall o \in O, c \in C, t \in T \tag{2}$$

The term on the left hand side of the equation indicates whether or not a truck of class c has to start at the construction site associated with order o at time t for the selected pattern a. If so, then this may be the first delivery for a truck of class c, which is captured by the first term on the right hand side of the equation, or it may be a truck of class c that has already made a delivery at some other construction site, which is captured in the second term on the right hand side of the equation.

We do not only have to ensure that the right vehicle arrives at a construction site at the right time for an order and its associated pattern, we also need to ensure that the vehicle remains at the construction site until it has finished unloading. Furthermore, in case an order requires special equipment, we have to ensure that the first vehicle to arrive, which brings the specialized equipment, remains until the last vehicle has finished its unloading operation. Therefore, we have to consider two slightly different situations, i.e., one for orders where no special equipment is required ($o \in O'$) and one for orders where special equipment is required ($o \in O''$). They will be discussed separately as the resulting constraints are slightly different.

Let U_o^c denote the time required to unload a vehicle of class c at the construction site associated with order o (a result of the vehicle's capacity and the unloading rate at the construction site). Vehicles do not necessarily have to leave immediately. (Recall that we assume that vehicles arrive just in time and any waiting time happens at the location from which the vehicle departs.) Therefore, we introduce variable $w_o^{c,t}$ to capture the number of vehicles of class c not unloading, but waiting at the construction site associated with order o at time t. The following set of constraints ensure that vehicles leave only when they are finished unloading and at the

same time ensure vehicle balance at every node of the time-space network:

$$\sum_{a \in A_o} P_{oa}^{c,t-U_o^c} z_{oa} + w_o^{c,t-1} = \sum_{o_1 \in O} x_{oo_1}^{ct} + v_o^{ct} + w_o^{ct} \quad \forall o \in O', c \in C, t \in T. \tag{3}$$

The first term on the left hand side captures vehicles that have just finished their unloading operation and the second term on the left hand side captures any vehicles waiting at the construction site. The right hand side captures that vehicles either move to another construction site, move to their home plant, or remain at the construction site.

In case an order requires special equipment for unloading, the situation is slightly more complicated. We have to ensure that the first vehicle that arrives remains at the site until the last vehicle leaves. To do so the terms on the left hand side of the equation need to be defined more carefully. In the first term, we need to make sure that the first vehicle cannot leave when it finishes unloading except in case this is the only unloading operation of the order, i.e., for a vehicle of class c, we need to exclude $t = start_{oa} + U_o^c$ if $c = first_{oa}$ unless $t = end_{oa}$. Furthermore, we need to add a term that ensures that the first vehicle can leave when all unloading operations have finished, i.e., we need to "release" a vehicle of class c at $t = end_{oa}$ when $c = first_{oa}$. This leads to the following set of constraints:

$$\sum_{\substack{a \in A_o: \\ \neg\{c = first_{oa} \wedge \\ t = start_{oa} + U_o^c \wedge \\ t \neq end_{oa}\}}} z_{oa} \cdot P_{oa}^{c,t-U_o^c} + \sum_{\substack{a \in A_o: \\ \{c = first_{oa} \wedge \\ t = end_{oa}\}}} z_{oa} \cdot P_{oa}^{c,start_{oa}} + w_o^{c,t-1} =$$

$$\sum_{o_1 \in O} x_{oo_1}^{ct} + v_o^{ct} + w_o^{ct} \quad \forall o \in O'', c \in C, t \in T. \tag{4}$$

The objective of the optimization consists of minimizing traveling time as well as gaps. Gaps might occur either as a consequence of delays between consecutive unloading operations scheduled at the same construction site or because of starting with the very first unloading operation after the end of the corresponding time window. An example of a gap between consecutive unloading operations is shown in Figure 2, where there is a delay between the second and third unloading operation.

As mentioned before, the objective is to fulfill all orders, to minimize the travel cost, and to avoid delays between two consecutive unloading operations for an order. Each fulfillment pattern has an associated amount of delay, so penalizing delays can easily be accomplished. Similarly, penalizing unfulfilled orders is also trivial. As the severity of these two undesirable aspects of a delivery schedule are quite different, we use different values for their penalty coefficients, namely β_1 and β_2. This leads to the following objective function, where p_c refers to the home plant of trucks of class c, t_{ij} represents the travel cost from location i to j, and $delay_{oa}$ represent the delay in fulfillment pattern a for order o:

$$\begin{aligned} \min \sum_{\substack{o\in O\\ c\in C\\ t\in T}} t_{p_c o} u_o^{ct} + \sum_{\substack{o_1,o_2\in O\\ c\in C\\ t\in T}} t_{o_1 o_2} x_{o_1 o_2}^{ct} + \sum_{\substack{o\in O\\ c\in C\\ t\in T}} t_{o p_c} v_o^{ct} + \\ \beta_1 \sum_{\substack{o\in O\\ a\in A_o}} delay_{oa} z_{oa} + \beta_2 \sum_{o\in O} (1 - \sum_{a\in A_o} z_{oa}). \end{aligned} \tag{5}$$

In order to ensure that vehicles start their daily tours from their home plants and return there after having executed their last unloading operation, the following constraints need to be added: the number of vehicles of a given class c available at a given plant p (n^{p_c}) equals the number of vehicles starting their tour plus the number of vehicles staying at the home plants throughout the day (s^{p_c}). Similarly, all vehicles need to return to their home plant:

$$n^{p_c} = s^{p_c} + \sum_o \sum_t u_o^{ct} \quad \forall c \tag{6}$$

$$n^{p_c} = s^{p_c} + \sum_o \sum_t v_o^{ct} \quad \forall c \tag{7}$$

Two different methods have been developed to generate a set of fulfillment patterns for the MCNF component.

The first myopic approach randomly generates a (relatively large) number of fulfillment patterns for each order. Recall that by definition a fulfilment pattern is feasible from an order's perspective, i.e., total demand is satisfied, unloading operations do not overlap, and special equipment is used when needed.

A single pattern is generated sequentially. First, the class of the vehicle that will perform the first unloading operation and the time at which the first unloading operation will start will be determined. Then, the class of the vehicle that will perform the second unloading operation and the time at which the second unloading operation will start will be determined. This process repeats until the total quantity that can be delivered by the vehicles exceeds the demand associated with the order. The probability of choosing a certain vehicle class is proportional to the number of vehicles within that class, taking into account special requirements for unloading equipment when selecting the first vehicle. The first vehicle should preferably start its unloading operation as close as possible to the beginning of the order time window. All subsequently scheduled vehicles should preferably start their unloading operation immediately after the previous unloading operation has finished. The probability of choosing a particular start time for an unloading operation is inversely proportional to the length of the delay caused by that start time. A set of x fulfillment patterns is generated in this way, making sure that all patterns generated for one order are unique.

Next, the set of patterns is expanded to a set of $10x$ patterns by adding delays to existing patterns, again randomly. A fulfillment pattern and an unloading operation of this pattern are randomly chosen. The sequence of vehicles in the pattern remains unchanged, but the start time of the selected unloading operation will be delayed by a small amount, again chosen randomly. The resulting pattern will be added to the pool of fulfillment patterns. The delays are added on purpose, because when resources are tight, in terms of fleet size, a delivery schedule with delays might be the only way to satisfy every order.

This approach has been chosen because it allows us to quickly generate a large number of possible patterns. However, it does not take into account the interaction between the various orders and their fulfillment patterns.

The second, more intelligent approach, does take the dependence between fulfillment patterns for different orders into account. Instead of randomly generating a large number of patterns, a

small set of compatible patterns will be generated. One by one, patterns are generated for each order in such a way that the set of patterns generated can be converted into a low-cost feasible delivery schedule. More specifically, a complete feasible delivery schedule is constructed greedily and the compatible patterns are derived from the delivery schedules for the individual orders. To construct a complete feasible delivery schedule, orders are processed in random order. For a given order, unloading operations are scheduled sequentially at the earliest possible start time with a vehicle with the largest possible capacity, taking into account any existing partial schedule for the vehicles. If more than one vehicle (with the same capacity) is available to perform the unloading operation, the vehicle is selected randomly with a bias inversely proportional to the distance of the home plant of the vehicle to the location where the unloading operation needs to take place. It is always possible to construct a feasible schedule this way, although the delays between consecutive unloading operations can sometimes be large.

The length of the planning horizon depends on the instance characteristics. It is set to the smallest value that accommodates all generated fulfillment patterns used during the initialization of the MCNF-component. If during execution patterns are generated that require a larger planning horizon, the length of the planning horizon is adjusted upward.

4 Variable Neighborhood Search Component

Any concrete delivery schedule may possibly be improved using methods inspired by VNS (see Mladenović and Hansen [15] and Hansen and Mladenović [8]). A high-level sketch of the basic steps of the implemented VNS can be found in Algorithm 1. The VNS stops after a given number of iterations, a given number of iterations in which no improvement is found, or when a time limit is reached.

During the *shaking phase* various neighborhoods are used to thoroughly explore the solution space and to avoid being trapped in a local optimum. The neighborhoods invert and exchange

Algorithm 1 Basic Steps of VNS

while stopping criterion not met **do** ▷ Iterations, Time

  $k \leftarrow 1$

  while $k \leq k_{max}$ **do**

    $x' \leftarrow$ `Shaking`(x, k)

    $x'' \leftarrow$ `Local Search`(x')

    if `accept`(x'') **then**

      $x \leftarrow x''$

      $k \leftarrow 1$ ▷ continue with first Neighborhood Structure

    else

      $k \leftarrow k + 1$ ▷ continue with next Neighborhood Structure

    end if

  end while

end while

the sequences of vehicles associated with the fulfillment of orders. A neighboring solution x' will be generated at random from neighborhood $N_k(x)$. The embedded *local search* (LS) improves every solution x' obtained after a shaking step. If the resulting solution, denoted by x'', is better than the current solution x, it will become the new current solution x and the shaking continues with the first neighborhood. If the incumbent solution x cannot be improved, the search continues with neighborhood $k+1$.

4.1 Shaking

In order to thoroughly explore the solution space - in terms of potential patterns - two shaking operators, resulting in six neighborhood structures have been designed and implemented.

The first shaking operator tries to replace sequences of vehicles in a pattern by vehicles *not being used* in any other pattern of the current delivery schedule. When selecting replacement vehicles there is a bias towards vehicles with large capacities and those whose home plant is located close to the construction site associated with the order (both characteristics tend to lead to low-cost schedules). The second shaking operator is similar, but this time the replacement vehicles are no longer limited to those not being used in the current solution. Rather replacement vehicles are selected randomly among all vehicles available.

Neighborhood structures N_k $(k = 1, 3, 5)$ relate to the first shaking operator and involve changes of up to $\frac{k+1}{2}$ patterns; neighborhood structures N_k $(k = 2, 4, 6)$ relate to the second shaking operator and involve changes of up to $\frac{k}{2}$ patterns. The patterns to be modified are going to be selected on a random basis. The selection probabilities are equiprobable. An overview on the set of neighborhood structures is given in Table 1. The position and the length of the sequence to be exchanged is determined randomly. In order to compensate for the loss of unloading capacity associated with the vehicles being removed, new vehicles will be inserted into the pattern until the demand of the corresponding order can be satisfied again. Furthermore, if the first vehicle of the sequence is being exchanged and the order requires special

Table 1: Set of Neighborhood Structures

κ	Shaking Operator	max number of patterns changed
1	`Replace_By_Unused`	1
2	`Replace_By_Any`	1
3	`Replace_By_Unused`	2
4	`Replace_By_Any`	2
5	`Replace_By_Unused`	3
6	`Replace_By_Any`	3

unloading equipment, only vehicles equipped with the right instrumentation will be considered when selecting the replacement for the first vehicle of the pattern.

After executing a shaking operator, the resulting patterns are feasible from the perspective of the orders, i.e., the demand can be satisfied, unloading operations do not overlap, and specialized unloading equipment is there when needed. However, the patterns do not constitute a feasible delivery schedule, as the movements of vehicles between plants and construction sites and movements of vehicles between constructions sites with intermediate stops at plants and the associated loading times are not considered. Shaking operators only affect the sequence of vehicles scheduled to unload at a construction site to satisfy demand. In order to truly evaluate the effects of a shaking operation, vehicle movements and thus the precise timing of unloading operations have to be considered. Two evaluation functions have been implemented to do so. Each takes a set of patterns, one for each order, and builds vehicle itineraries compatible with these patterns, which imply start times of the unloading operations performed at the constructions sites. The resulting solution, a complete delivery schedule, will not only be feasible from the orders' point of view, but also from the vehicles' point of view, i.e., it will properly account for driving time and loading time between consecutive unloading operations. The two evaluation

functions will be described in more detail in the next section.

4.2 Evaluation Functions

As the shaking operators and the LS operators employed by the VNS perturb the patterns of a solution x, the movements of vehicles and the timing of unloading operations need to be re-computed. Two heuristic approaches have been developed to do so: *forward time setting* and *backward time setting*. They are executed consecutively, starting with forward time setting. The delivery schedule with the lowest cost is chosen. The procedures have been inspired by the Critical Path Method used in activity planning (see Moder [16]). However, modifications were necessary to handle vehicles with special unloading equipment, as they need to remain at the construction site to assist with concrete delivery of vehicles arriving later. If not handled carefully, deadlock situations might arise.

A deadlock situation arises in the following situation (see Figure 4). We need to set the

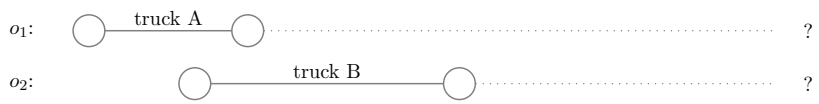

Figure 4: Deadlock situation

time for the unloading operations associated with orders o_1 and o_2. Each order requires two deliveries and each order requires special unloading equipment. Vehicle A has to perform the first unloading operation of order o_1 and vehicle B has to perform the second unloading operation. For order o_2, the sequence is reversed, i.e., vehicle B has to perform the first unloading operation and vehicle A has to perform the second unloading operation. Observe that when the time of the first unloading operation is set, it is not yet known how long the vehicle needs to stay at the construction site, because the remaining unloading operations have not been scheduled yet. If the forward time setting heuristic sets the time of the first unloading operation of order o_1

followed by the time of the first unloading operation of order o_2, we are deadlocked. Vehicle A, which has been scheduled for the first unloading operation of order o_1 cannot be released until the second unloading operation has been performed by vehicle B. However, vehicle B is tied up with order o_2.

In order to avoid deadlock situations, the scheduling of certain unloading operations has to be delayed artificially. The resulting schedule can be improved by applying backward time setting afterwards. However finding an improvement is not guaranteed, therefore the best solution obtained by any of the two procedures will be accepted.

A detailed description of the forward time setting algorithm can be found in Algorithm 2. The forward time setting algorithm schedules unloading operations in order of their earliest possible start time, which are updated dynamically; ties are broken arbitrarily. Unloading operations are scheduled as early as possible. The first unloading operation of an order cannot start before the beginning of its associated time window. When the first unloading operation of an order o requiring special unloading equipment is scheduled, other orders o^* requiring the same vehicle are temporarily blocked. The block is removed as soon as all the unloading operations of order o have been scheduled.

Backward time setting proceeds similarly. The backward time setting algorithm schedules unloading operations in order of their latest start times, which are updated dynamically. The "latest" start time of the last operation of an order is set to the start time in the schedule produced by the forward time setting algorithm.

4.3 Local Search

Three operators have been implemented for the LS. One of the three LS operators (`Shrink`) is specifically designed for this application, the other two (`Intra Pattern Move` and `Inter Pattern Swap`) are similar to well-known LS operators for vehicle routing problems. (See Gendreau et al. [6] and Kindervater and Savelsbergh [11] for a discussion of LS for vehicle routing

Algorithm 2 Forward Time Setting

```
1: Initialize ordered list L with first unloading operations of orders
2: while L ≠ ∅ do
3:     l ← first(L)
4:     L ← L − l
5:     o ← l.order                                   ▷ order under consideration
6:     k ← l.vehicle                                 ▷ vehicle to perform unloading operation
7:     t ← l.time                                    ▷ start time of unloading operation
8:     if order o is not blocked then
9:         if vehicle k can reach construction site of o by time t then
10:            schedule unloading operation with vehicle k at time t for order o
11:            if there are remaining unloading operations for order o then
12:                Update L                          ▷ insert next unloading operation into list
13:            end if
14:            if first unloading operation of order requiring special equipment then
15:                for o* ∈ O requiring special equipment and not yet started do
16:                    block o* because of o
17:                end for
18:            end if
19:            if last unloading operation of order requiring special equipment then
20:                for orders o* that have been blocked by order o do
21:                    unblock o*
22:                    if order o* no longer blocked by any order then
23:                        Update L                  ▷ insert first unloading operation of o* into list
24:                    end if
25:                end for
26:            end if
27:        else
28:            Update L                              ▷ insert unloading operation next possible instance of time
29:        end if
30:    end if
31: end while
```

problems.)

All operators are executed based on first improvement. The `Intra Pattern Move` operator removes a single unloading operation of the pattern of a given order and inserts it at all other possible positions in that pattern. The `Inter Pattern Swap` operator exchanges two unloading operations of patterns of two different orders. The `Shrink` operator tries to remove unnecessary unloading operations from a pattern as the quantity of concrete ordered might already be satisfied by the other unloading operations.

During the execution of the three LS operations the timing of unloading operations will be ignored. The actual move, swap, or deletion takes place, taking only the feasibility of the sequence of unloading operations into account from the perspective of the order. Afterwards the timing of the unloading operations will be determined by the two evaluation functions (forward time setting and backward time setting).

A high-level description of the LS can be found in Algorithm 3. The `Shrink` operator is applied to eliminate any unnecessary unloading operations from the patterns. Next, the `Intra Pattern Move` operator is executed on a first improvement basis followed by the `Inter Pattern Swap` operator as soon as the `Intra Pattern Move` is unable to improve the solution, again on a first improvement basis. To complete the LS, the `Shrink` operator is applied one more time. The three operators are described in more detail below.

`Shrink`: This operator ensures that no pattern has any unnecessary unloading operations. It may happen that the demand of an order can be satisfied with fewer unloading operations. Each unloading operation is removed from the pattern to see if the resulting pattern is still feasible (from an order's perspective), i.e., demand is satisfied and specialized unloading equipment is there when needed. After an unloading operation is deleted, feasibility is re-established using the evaluation functions.

`Intra Pattern Move`: This operator ensures that no lower cost delivery schedule can be obtained by changing the position of one unloading operation within a pattern. This is im-

Algorithm 3 Local Search(x)

$x = \texttt{Shrink}(x)$ ▷ Shrink

$improved \leftarrow true$

while $improved$ **do**

  $x' \leftarrow \texttt{Intra Pattern Move}(x)$ ▷ Intra Pattern Move

  if $\texttt{accept}(x')$ **then**

    $x \leftarrow x'$

    $improved \leftarrow true$

  else

    $x' = \texttt{Inter Pattern Swap}(x)$ ▷ Inter Pattern Swap

    if $\texttt{accept}(x')$ **then**

      $x \leftarrow x'$

      $improved \leftarrow true$

    else

      $improved \leftarrow false$

    end if

  end if

end while

$x = \texttt{Shrink}(x)$ ▷ Shrink again

plemented by simply trying all other positions within a pattern for each unloading operation. Feasibility needs to be re-established using the evaluation function for every attempted change. Of course, any requirements concerning specific unloading equipment will be taken into account.

`Inter Pattern Swap`: This operator ensures that no lower cost delivery schedule can be obtained by exchanging two unloading operations for two different orders. This is implemented by simply trying all possible exchanges of two unloading operations for every combination of two orders. Feasibility needs to be re-established using the evaluation function for every attempted exchange. Of course, any requirements concerning specific unloading equipment will be taken into account.

5 Hybrid Approach

It is impractical, if not impossible, to generate all possible fulfillment patterns and have the integer MCNF component select one pattern for each order and, at the same time, determine vehicle itineraries in such a way that the resulting costs of vehicle movements are minimum. A more pragmatic approach is to generate a small number of fulfillment patterns and have the integer MCNF component optimize over these patterns. Further, if the solution thus obtained is not satisfactory, additional fulfillment patterns can be generated and another optimization can be performed. In such a scheme, the performance depends critically on the way patterns are generated. This is where our approach is novel and innovative, as the generation of fulfillment patterns will be done using VNS.

As the VNS produces complete delivery schedules, the fulfillment patterns transferred from the VNS component to the MCNF component will be feasible from an order's perspective, i.e., the demand for concrete can be completely satisfied, consecutive unloading operations do not overlap, and special unloading equipment is available when needed. The MCNF component identifies among all the patterns in its "database" of patterns a fulfillment pattern for each

order and a set of associated feasible vehicle itineraries, i.e., with enough time between unloading operations to drive from one location to a plant, fully load the vehicle with concrete, and drive to the next location, that minimizes total costs, i.e., direct costs (travel costs) and indirect costs (delays between consecutive deliveries).

A graphical representation of the hybrid approach can be found in Figure 5. The first step is to generate an initial set of fulfillment patterns for each order. After solving the resulting integer MCNF problem, the solution is transferred to the VNS component, which locally improves the solution and in the process generates additional fulfillment patterns. The exploration of the search space is guided by the characteristics of the orders as well as by the fulfillment patterns of the solution produced by the MCNF component.

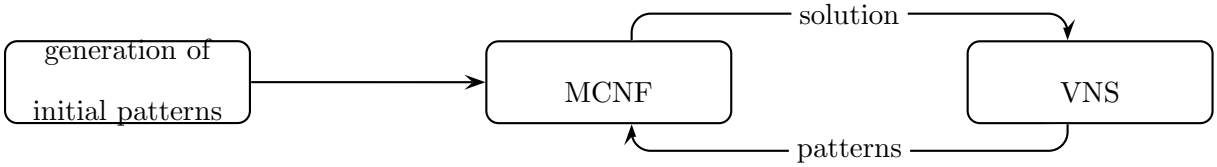

Figure 5: Hybrid Solution Procedure

While the VNS component searches the solution space, it encounters, typically, one or more improved solutions. After a certain period of time, all the patterns in these improving solutions are collected and transferred to the MCNF component. Therefore, the database of fulfillment patterns kept by the MCNF component keeps growing and the chances for being able to find a set of compatible patterns for the orders resulting in low-cost delivery schedules increases. The hybrid approach iterates between the two components, each time transferring the best delivery schedule from the MCNF component to the VNS component and the fulfillment patterns of the solutions encountered by the VNS component to the MCNF component. Another benefit of the collaboration is that if the set of initial fulfillment patterns does not allow the MCNF component to produce a complete delivery schedule covering every order, the VNS component

is quickly able to identify "missing patterns" and establish a complete solution.

In the actual implementation, we have chosen to set things up as follows. The solution process is controlled by a few parameters: the run time limit, the number of iterations n_{iter}, and the allocation of time between MCNF and VNS per iteration. The total run time is divided equally over the iterations. In each iteration, a fraction p ($0 \leq p \leq 1$) of the time per iteration is allocated to the MCNF component and the remainder to the VNS component.

We have to consider the possibility that the MCNF component is unable to find an integer solution in the first iteration or is unable to find an improving integer solution in subsequent iterations due to the imposed time limit. If this happens in the first iteration, then a feasible solution is randomly created using the current pool of patterns and the VNS component starts from there. If this happens in subsequent iterations, then the best incumbent solution of the previous VNS will be used as an initial solution.

6 Computational Experiments

6.1 Data Description

For our computational experiments, we use real-life data from a concrete company located in northern Italy. The company provided us with all orders placed between January and November of 2006. On average 51.8 orders had to be served per day and an average quantity of 616.55 cubic meters (m^3) of concrete had to be delivered per day. On average 41.6% of orders do not require specialized unloading equipment, 55.6% of orders require a pump, and 2.8% require a conveyor belt. The company's fleet consists of 33 vehicles, 14 of which can deliver concrete, two of which are equipped with specialized unloading instrumentation and cannot deliver concrete, and 17 of which are multi-functional in that they are equipped with specialized unloading instrumentation but can also deliver concrete. The average capacity of the vehicles is 8.6 m^3.

We tested the various algorithms on a set of 15 instances. Each instance represents one

day, i.e., all orders that have to be satisfied on that particular day. The instances have different characteristics and can be grouped into three categories: small, medium, and large. An overview of the characteristics of the instances and the three categories can be found in Table 2. The number of orders is denoted by n_o, the total quantity of concrete to be delivered is denoted by $sumQ$, the average, smallest, and largest order quantity are denoted by $avgQ$, Q_{min}, and Q_{max} respectively. The standard deviation of the order quantities is denoted by σ_Q. Instances with fewer than 50 orders are considered small, instances with between 50 and 60 orders are considered medium-sized, and instances with more than 60 orders are considered large. Averages over all instances in each category are also provided.

For the MCNF-component, the parameters β_1 and β_2 used in the objective function to penalize delays and unfulfilled orders, respectively, were set to 3 and 5000.

All computational experiments have been conducted on a desktop PC with a 3.2 GHz Intel Pentium©processor and 3 Gb of RAM. The Xpress-Optimizer (version 2006B) was used for solving integer MCNF problems. Run times are given in seconds.

6.2 Integer Multi-Commodity Flow Formulation Initialization

Two methods have been discussed for generating fulfillment patterns for the MCNF component in Section 3. The first randomly generates fulfillment patters for each order; the second generates fewer, but compatible fulfillment patterns for the orders.

To evaluate the impact of the set of fulfillment patterns on the performance of the MCNF component, we compared the following settings: 100 randomly generated fulfillment patterns per order versus 20 intelligently generated fulfillment patterns per order. The results are presented Table 3. Each entry in the table represents the average over five runs.

The best solution value found among the five runs is denoted by z_{min}. The average of the five solution values is denoted by z_{avg}. The average of the five run times (in seconds) is denoted by t_{avg}. In the last two columns, we present the difference between the two variants in terms of

average solution values ($\%z_{gap}$) and total run times ($\%t_{gap}$).

The results clearly demonstrate that intelligently generating fulfillment patterns, as opposed to generating them randomly, dramatically improves solution quality and performance. This observation, in a way, already signals that our hybrid approach, which uses VNS to identify high-quality compatible fulfillment patterns makes sense.

The increase in run time when fulfillment patterns are generated intelligently is not the result of an increase in pattern generation time. Intelligently generating 20 patterns takes only 1.8 seconds for the largest instance. Using the brute force approach to generate 100 patterns for the same instance takes 0.13 seconds. The increase in run time is due to the fact that selecting patterns by means of the MCNF takes more time. We want to point out that the run times reported in Table 3 represent the time required to prove optimality. In the hybrid approach, however, we rarely solve problems to proven optimality, as we impose run time limits for the solution process. When the run time limit is reached, the best known solution will be used as an input for VNS.

6.3 The Hybrid Approach

Two parameters control the execution of our hybrid approach: the number of iterations n_{iter}, i.e., the number of times we executed the embedded MCNF and VNS components, and the fraction of time p allocated for the execution of the embedded MCNF component (and thus the fraction of time $1 - p$ allocated for the execution of the embedded VNS component) in each iteration. The total run time is distributed equally over all iterations. Within a single iteration at most $p\%$ of the designated run time will be allocated to MCNF. The remaining time will be allocated to VNS. The best solution found will be used as an input for VNS.

Based on initial experimentation, the following settings were chosen for the hybrid approach for the sets of experiments in which we compare the performance of the hybrid approach to the performance of the MCNF component, the VNS component, and the commercially avail-

able solution based on simulated annealing (SA). The hybrid approach starts with a set of 15 intelligently generated fulfillment patterns per order. The number of iterations n_{iter} is 8, i.e., we switch back and forth between the MCNF component and the VNS component 8 times. In each iteration, 30% of the run time is allocated to the MCNF and, therefore, 70% of the run time is allocated to the VNS component.

To understand and analyze the sensitivity of the hybrid approach with respect to these parameters, we conducted the following experiment. For the medium-sized instances and for a run time limit of 600 and 1200 seconds, we varied the number of iterations n_{iter} between 6 and 10 (in steps of two) and the percentage of time allocated to the embedded MCNF component from 20 to 40% (in steps of 10). The results are reported in Table 4. For every combination of parameters, each instance was solved five times (instantiated with a different random seed); values in the table represent averages over all instances.

First and foremost, the results show that the hybrid approach is quite robust and not very sensitive to the chosen parameters. Different parameter choices lead only to small variations in the average solution values obtained. Second, the results also validate, to some extend, our intuition that a certain amount of time has to be allocated to MCNF component, almost independent of the instances being solved. With a run time limit of 600 seconds, it is better to allocated 40% of the time per iteration to the MCNF component, whereas with a run time limit of 1200 seconds it suffices to allocate 20% of the time to the MCNF component. Similarly, with a run time limit of 600 seconds, it is better to have only 6 iterations, whereas with a run time limit of 1200 seconds, the number of iterations does not seem to have a major impact anymore.

As mentioned above, most of the experiments focus on investigating the efficacy of the proposed hybrid approach. We do so by comparing its performance to the performance of the MCNF component, the performance of the VNS component, and the commercially available solution based on SA.

6.4 Hybrid Approach vs. Integer Multi-Commodity Network Flow

The main difference between the hybrid approach and the MCNF component is the set of fulfillment patterns used during the optimization. The hybrid approach relies heavily on the VNS to provide good sets of fulfillment patterns, whereas the pure MCNF component relies on a greedy approach for generating compatible patterns. To assess the impact of the different schemes for generating patterns, we conducted the following experiment. After executing the hybrid approach for a given maximum run time, we look at the number of fulfillment patterns in the pool of patterns at termination. Next, we initialize the MCNF component with the same number of patterns, using greedy construction of compatible patterns, and allow it to run for the same amount of time. The results are found in Table 5.

The values presented for a category correspond to averages over the values obtained for all instances in that category. For both the MCNF component and the hybrid approach, we report the best and the average solution value, denoted by z_{min} and z_{avg}, respectively. The overall best solution value is highlighted in bold. The percentage improvement of the average solution value obtained with the hybrid approach over the average solution value obtained with the MCNF component is presented in the last column (with heading $\%z_{gap}$). The average number of patterns generated is denoted by $pats$. To analyze and understand the impact of maximum run time on both approaches, the maximum run time t_{max} was varied between 150 and 4800 seconds. Variations in the results produced by the MCNF component are the result of different sets of initial fulfillment patterns.

The results demonstrate the superiority of the hybrid approach. The best solution is *always* obtained by the hybrid approach. Even the average solution value of the hybrid approach is almost always better than the best solution value obtained by the pure MCNF component. The VNS component in the hybrid approach is able to intelligently diversify and add high-quality fulfillment patterns to the pool of patterns, enabling the MCNF component to find low-cost

delivery schedules.

6.5 Hybrid Approach vs. Variable Neighborhood Search

The main difference between the hybrid approach and the VNS component is that the hybrid approach, by means of the MCNF component, at various time during the solution process, is able to take a global view and consider all the fulfillment patterns encountered during the search and combine them in a cost-effective way, thereby moving to a promising part of the search space.

As in the previous set of experiments, we run both algorithms for a maximum of t_{max} seconds and compare the results, where the VNS is initialized with one fulfillment pattern per order. The results are presented in Table 6. As before, the values presented for a category correspond to averages over the values obtained for all instances in that category. For both the hybrid approach and the VNS, we report the best and the average solution value, denoted by z_{min} and z_{avg}, respectively. The overall best solution value is highlighted in bold. The percentage improvement of the average solution value obtained with the hybrid approach over the average solution value obtained with the VNS is presented in the last column (with heading $\%z_{gap}$).

Again, the results demonstrate the superiority of the hybrid approach. The best solution is *always* obtained by the hybrid approach. Even the average solution value of the hybrid approach is almost always better than the best solution value obtained by the pure VNS component. The MCNF component in the hybrid approach is able to diversify the search effectively enabling the VNS component to quickly reach areas of the search space containing low-cost delivery schedules.

We observe that both methods, not surprisingly, produce better solutions if given more time. The hybrid approach starts to produce high-quality solutions much more quickly, already when given about 600 seconds (maybe a little more for large instances). When given 600 seconds the quality of solutions produced by the hybrid approach and the VNS differs markedly, a difference of around 17% for medium-size and 25% for large instances. With more time, the VNS is able to reduce the difference in quality. Our initial contention that the MCNF component may provide

a powerful diversification mechanism for the VNS seems to be validated here. The integration of both methods allows us to reach a high-quality solution much more quickly, as opposed to when relying only on the mechanisms embedded in the VNS itself.

For run time limits of 300 and 600, we provide a more detailed overview of the results in Table 7.

The results reinforce our earlier interpretations. The variance in quality of the solutions produced by the hybrid approach is much smaller than for the VNS with these run time limits, especially when we examine the results for the run time limit of 600 seconds. This demonstrates once more the ability of the MCNF component, with its global view, to quickly move to a good part of the solution space. The VNS may reach that same part of the search space at some point, but it will require much longer to do so.

6.6 Hybrid Method vs. Simulated Annealing

Finally, we compare the hybrid approach to a commercial solution specifically developed for this type of ready-mixed concrete delivery scheduling. The commercial solution relies on a simulated annealing (SA) algorithm (see Kirkpatrick et al. [12] for an introduction to SA). We compare the delivery schedules obtained by our hybrid approach to those produced by the commercial solution when given a maximum run time of 150 seconds, which is the typical time required by the SA-based approach to converge to its final solution. The results are presented in Table 8. The results obtained by our hybrid approach represent average values over 5 runs. The results obtained by the commercial solution represent average values over 25 runs (the number of cooling phases was varied between 1 and 5, with 5 runs for each variant).

The best solution for each instance is always found by our hybrid approach. For small instances the average solution quality can be increased by 38.34%, for medium-size instances

the average solution quality can be improved by 37.88%, and for large instance the average solution quality can be improved by 38.09%.

7 Conclusion and Final Remarks

Our study demonstrates the potential of integrating optimization and heuristic techniques. Our hybrid approach is extremely effective, especially when high-quality delivery schedules need to be produced in a relatively short amount of time.

The MCNF component takes a *global* view and selects the best set of fulfillment patterns for the orders from among a set of potential fulfillment patters. The VNS component takes *a local* view and generates delivery schedules in the neighborhood of the delivery schedule produced by the MCNF component, thus constructing additional fulfillment patterns and diversifying the search. The delivery schedules found by the VNS component enrich the pool of fulfillment patterns of the MCNF component, thereby enabling it to finding better solutions. The hybrid approach iterates between the global view and the local view. The set of fulfillment patterns is constantly updated as the embedded VNS component continually adds fulfillment patterns.

Environments in which high-quality solutions to complex problems need to be produced in a short amount of time can be found everywhere, and these environments seem to be especially suited for hybrid approaches. The hybridization enables us to combine the strengths of optimization and heuristic techniques and to compensate for their weaknesses. Heuristic techniques efficiently and effectively exploring the solution space around a known feasible solution. Optimization techniques take a global view and can to jump to promising parts of the solution space. The combination can be extremely powerful.

The computational efficiency of our hybrid approach may be further enhanced, if necessary, by parallelizing the proposed scheme. Multiple VNS components, initialized with different random seeds, can be run simultaneously to generate patterns for the MCNF component. Similarly,

multiple MCNF components can be employed, each seeded with different sets of initial fulfillment patters. It is even possible to consider schemes in which a centralized database of patterns and feasible delivery schedules is envisioned and MCNF components and VNS components ask for sets of patterns and a feasible delivery schedule from the central repository. This is a topic of potential future research.

Acknowledgements

We would like to thank Hans Karl Huber and Daniela Feichter from Rienz Beton for providing us access to real-world data. Martin Savelsbergh was supported, in part, by AFOSR grant 240664R. Financial support from the Austrian Science Fund (FWF) under grant P20342-N13 is gratefully acknowledged.

Table 2: Properties of selected Instances

testcase	n_o	$sumQ$	$avgQ$	Q_{min}	Q_{max}	σ_Q
1	27	554.5	20.54	0.5	97.5	28.92
2	28	305.75	10.92	0.75	48	12.42
3	33	413	12.52	0.5	101.5	19.52
4	34	535	15.74	0.5	98	19.67
5	39	498.5	12.78	1	178	29.71
small	32.2	461.35	14.50	0.65	104.6	22.05
6	50	736	14.72	0.5	172	30.15
7	50	502	10.04	0.5	48	11.05
8	55	491	8.93	1	36	8.67
9	55	824.5	14.99	1	104	21.15
10	60	648	10.80	0.25	66	15.18
medium	54	640.3	11.90	0.65	85.2	17.24
11	65	776	11.94	0.5	133.5	21.10
12	65	637.75	9.81	0.25	55	10.25
13	70	719	10.27	0.5	53	11.71
14	70	886	12.66	0.5	99	16.66
15	76	721.25	9.49	0.25	115	14.36
large	69.2	748	10.83	0.4	91.1	14.82

Table 3: Initial Base Patterns: Random vs. Compatible Patterns

	BF			IBP			IPB vs BF	
testcase	z_{min}	z_{avg}	t_{avg}	z_{min}	z_{avg}	t_{avg}	$\%z_{gap}$	$\%t_{gap}$
1	2982.80	6151.30	74.23	**2168.05**	2236.54	118.29	-63.64%	59.35%
2	1407.12	1451.65	7.22	**1244.38**	1276.04	11.89	-12.10%	64.68%
3	2108.20	2178.06	12.41	**1922.65**	1947.91	22.06	-10.57%	77.83%
4	2284.02	2427.55	23.13	**2055.27**	2131.55	104.31	-12.19%	350.93%
5	2813.27	3047.74	30.22	**2480.60**	2519.20	59.97	-17.34%	98.45%
small	2319.08	3051.26	29.44	**1974.19**	2022.25	63.30	-23.17%	130.25%
6	3684.87	3873.28	371.87	**3136.22**	3173.99	1320.30	-18.05%	255.05%
7	2572.33	2645.08	33.03	**2376.78**	2506.69	279.25	-5.23%	745.35%
8	2501.37	2643.80	33.25	**2437.12**	2478.77	135.90	-6.24%	308.68%
9	4192.85	7514.34	3359.29	**3237.23**	3287.30	8064.79	-56.25%	140.07%
10	3249.75	5901.09	854.11	**2739.30**	2834.30	2003.82	-51.97%	134.61%
medium	3240.23	4515.52	930.31	**2785.33**	2856.21	2360.81	-27.55%	316.75%
11	3401.10	3520.92	1828.77	**2812.85**	2944.49	1802.65	-16.37%	-1.43%
12	3024.80	3128.63	259.18	**2652.65**	2767.51	1043.74	-11.54%	302.71%
13	3318.97	3535.87	557.42	**3179.18**	3260.93	3508.40	-7.78%	529.40%
14	3966.80	4133.45	5741.05	**3512.55**	3598.98	24195.52	-12.93%	321.45%
15	3110.68	3226.02	352.35	**2956.88**	2992.23	1715.96	-7.25%	387.00%
large	3364.47	3508.98	1747.75	**3022.82**	3112.83	6453.25	-11.17%	307.83%
all	2998.96	3680.49	955.33	**2620.91**	2691.83	3177.51	-20.04%	255.12%

Table 4: Parameter Study

		$t_{max} = 600$				$t_{max} = 1200$			
n_{iter}	p	20%	30%	40%	avg	20%	30%	40%	avg
6		2354.25	2311.94	2331.90	2332.70	2266.41	2285.19	2318.37	2289.99
8		2355.00	2364.59	2320.95	2346.84	2281.67	2289.56	2299.03	2290.09
10		2367.64	2353.87	2339.53	2353.68	2309.02	2286.79	2281.53	2292.45
avg		2358.96	2343.47	2330.79		2285.70	2287.18	2299.64	

Table 5: Hybrid vs. MCNF (based on equal max. run times, classwise comparison)

		MCNF		Hybrid			
class	t_{max}	z_{min}	z_{avg}	z_{min}	z_{avg}	$\%z_{gap}$	$pats$
small	150	1900.55	2151.01	**1597.10**	1674.84	-22.14%	28.00
	300	1870.40	1944.18	**1575.05**	1627.38	-16.29%	32.00
	600	1868.81	1924.70	**1558.61**	1605.30	-16.59%	33.00
	1200	1852.15	1909.95	**1560.06**	1600.45	-16.2%	35.60
	2400	1827.22	1874.83	**1535.54**	1570.12	-16.25%	41.40
	4800	1806.94	1859.73	**1546.52**	1578.15	-15.14%	45.40
medium	150	2769.93	4190.80	**2550.09**	2729.95	-34.86%	24.40
	300	2786.29	3606.75	**2322.64**	2457.30	-31.87%	27.20
	600	2717.86	3355.17	**2268.40**	2364.59	-29.52%	30.20
	1200	2641.14	2940.63	**2207.48**	2289.56	-22.14%	34.80
	2400	2580.30	2739.30	**2190.75**	2260.75	-17.47%	36.00
	4800	2599.96	2689.42	**2164.66**	2245.08	-16.52%	38.40
large	150	3083.38	3765.57	**2922.04**	3195.76	-15.13%	22.00
	300	2987.91	3692.10	**2732.46**	2842.27	-23.02%	24.40
	600	2897.71	3044.75	**2465.38**	2612.09	-14.21%	28.20
	1200	2857.92	2974.79	**2399.34**	2500.48	-15.94%	31.60
	2400	2840.36	2951.44	**2408.09**	2493.39	-15.52%	32.60
	4800	2817.14	2922.89	**2375.71**	2461.54	-15.78%	35.20

Table 6: Hybrid vs. VNS (based on equal run times, classwise comparison)

		VNS		Hybrid		
class	t_{max}	z_{min}	z_{avg}	z_{min}	z_{avg}	$\%z_{gap}$
small	150	1679.94	1796.89	**1597.10**	1674.84	-6.79%
	300	1584.93	1683.56	**1575.05**	1627.38	-3.34%
	600	1578.29	1644.61	**1558.61**	1605.30	-2.39%
	1200	1573.76	1624.99	**1560.06**	1600.45	-1.51%
	2400	1568.32	1611.83	**1535.54**	1570.12	-2.59%
	4800	1568.32	1605.32	**1546.52**	1578.15	-1.69%
medium	150	3341.22	3767.34	**2550.09**	2729.95	-27.54%
	300	2875.96	3224.47	**2322.64**	2457.30	-23.79%
	600	2475.64	2833.09	**2268.40**	2364.59	-16.54%
	1200	2335.47	2477.99	**2207.48**	2289.56	-7.60%
	2400	2282.88	2369.67	**2190.75**	2260.20	-4.62%
	4800	2257.36	2345.65	**2164.66**	2242.28	-4.41%
large	150	4035.64	4501.99	**2922.04**	3195.76	-29.01%
	300	3528.55	3893.28	**2732.46**	2842.27	-27.00%
	600	3182.03	3471.21	**2465.38**	2612.09	-24.75%
	1200	2691.74	2876.55	**2399.34**	2500.48	-13.07%
	2400	2556.84	2667.09	**2408.09**	2493.39	-6.51%
	4800	2534.85	2619.46	**2375.71**	2461.54	-6.03%

Table 7: Hybrid vs. VNS

	runtime = 300 secs					runtime = 600 secs				
	vns		hybrid			vns		hybrid		
	z_{min}	z_{avg}	z_{min}	z_{avg}	$\%z_{gap}$	z_{min}	z_{avg}	z_{min}	z_{avg}	$\%z_{gap}$
1	1590.83	1673.09	**1581.68**	1639.95	-1.98%	1589.95	1642.54	**1510.13**	1585.68	-3.46%
2	1161.83	1228.30	**1118.00**	1140.30	-7.16%	1161.83	1226.03	**1131.77**	1144.55	-6.65%
3	**1534.13**	1600.29	1541.45	1585.43	-0.93%	1534.13	1588.07	**1510.75**	1550.60	-2.36%
4	1556.23	1689.96	**1547.87**	1615.70	-4.39%	1547.97	1627.58	**1543.00**	1616.30	-0.69%
5	**2081.63**	2226.19	2086.23	2155.51	-3.17%	**2057.55**	2138.81	2097.42	2129.38	-0.44%
small	1584.93	1683.56	**1575.05**	1627.38	-3.53%	1578.29	1644.61	**1558.61**	1605.30	-2.72%
6	3185.18	3502.68	**2632.18**	2748.89	-21.52%	2628.63	2836.28	**2601.60**	2688.99	-5.19%
7	2127.08	2436.70	**1953.87**	2071.51	-14.99%	2056.60	2177.31	**1926.78**	2034.41	-6.56%
8	2040.92	2191.52	**2025.88**	2059.70	-6.02%	2040.92	2104.50	**2022.67**	2050.10	-2.58%
9	3705.90	4256.65	**2784.83**	3025.02	-28.93%	3201.90	3814.48	**2634.17**	2834.58	-25.69%
10	3320.70	3734.78	**2216.43**	2381.39	-36.24%	2450.13	3232.86	**2156.77**	2214.86	-31.49%
medium	2875.96	3224.47	**2322.64**	2457.30	-21.54%	2475.64	2833.09	**2268.40**	2364.59	-14.30%
11	3367.17	3986.32	**2616.25**	2728.93	-31.54%	3052.72	3353.09	**2277.22**	2440.06	-27.23%
12	3224.45	3492.57	**2312.23**	2409.69	-31.01%	2736.68	3100.87	**2215.73**	2323.32	-25.08%
13	3483.98	3816.04	**2849.75**	2958.78	-22.46%	3038.58	3442.29	**2523.17**	2705.24	-21.41%
14	4206.72	4366.80	**3347.53**	3463.85	-20.68%	3854.63	3978.70	**2939.25**	3094.95	-22.21%
15	3360.42	3804.66	**2536.55**	2650.09	-30.35%	3227.52	3481.08	**2371.52**	2496.88	-28.27%
large	3528.55	3893.28	**2732.46**	2842.27	-27.21%	3182.03	3471.21	**2465.38**	2612.09	-24.84%
total	2663.14	2933.77	**2210.05**	2308.98	-17.42%	2411.98	2649.63	**2097.46**	2193.99	-13.95%

Table 8: Hybrid vs. SA: Comparion of solution quality and run time

	Hybrid			SA				
testcase	z_{min}	z_{avg}	t_{avg}	z_{min}	z_{avg}	t_{avg}	$\%z_{gap}$	$\%t_{gap}$
1	**1599.85**	1732.60	157.13	2337.61	2880.06	155.56	-39.84%	1.01%
2	**1139.53**	1157.88	155.37	1429.77	1672.89	137.64	-30.79%	12.88%
3	**1547.82**	1578.51	158.34	2077.83	2464.71	163.96	-35.96%	-3.43%
4	**1575.98**	1654.95	159.52	2409.88	2921.99	177.44	-43.36%	-10.10%
5	**2122.33**	2250.26	163.20	3223.41	3863.32	202.84	-41.75%	-19.54%
small	**1597.10**	1674.84	158.71	2295.70	2760.59	167.49	-38.34%	-3.84%
6	**2698.88**	2909.58	168.25	4057.40	4593.99	218.48	-36.67%	-22.99%
7	**2052.37**	2193.88	168.72	3110.75	3866.32	170.16	-43.26%	-0.85%
8	**2100.62**	2178.18	170.71	3231.78	3781.86	185.52	-42.40%	-7.98%
9	**3333.25**	3641.25	177.11	4215.76	5135.75	231.60	-29.10%	-23.53%
10	**2565.32**	2726.87	178.11	3670.80	4396.75	203.92	-37.98%	-12.66%
medium	**2550.09**	2729.95	172.58	3657.30	4354.93	201.94	-37.88%	-13.60%
11	**2673.67**	2984.90	183.61	3810.40	4673.39	215.08	-36.13%	-14.63%
12	**2444.65**	2692.13	177.17	4016.47	4510.84	207.92	-40.32%	-14.79%
13	**3123.72**	3410.02	185.99	4654.08	5587.77	211.20	-38.97%	-11.94%
14	**3499.70**	3734.63	190.76	5347.55	6027.76	245.16	-38.04%	-22.19%
15	**2868.47**	3157.13	191.27	4358.41	5008.30	204.68	-36.96%	-6.55%
large	**2922.04**	3195.76	185.76	4437.38	5161.61	216.81	-38.09%	-14.02%
all	**2391.76**	2574.91	173.19	3524.33	4159.21	196.75	-38.10%	-10.71%

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[2] L. Asbach, U. Dorndorf, and E. Pesch. Analysis, modeling and solution of the concrete delivery problem. *European Journal of Operational Research*, to appear.

[3] O. Bräysy. A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS Journal of Computing*, 15(4):347 – 368, 2003.

[4] R. De Franceschi, M. Fischetti, and P. Toth. A new ilp-based refinement heuristic for vehicle routing problems. *Mathematical Programming*, 105:471–499, 2006.

[5] M. T. Durbin. *The Dance of the Thirty-Ton Trucks: Demand Dispatching in a Dynamic Environment*. PhD thesis, George Mason University, 2003.

[6] M. Gendreau, G. Laporte, and J.Y. Potvin. Vehicle routing: Modern heuristics. In E. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*. John Wiley & Sons, New York (NY), USA, 1997.

[7] F.W. Glover, D. Klingman, and N.V. Phillips. *Network Models in Optimization and their Application in Practice*. John Wiley & Sons, New York (NY), USA, 2003.

[8] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[9] V.C. Hemmelmayr, K.F. Doerner, and R.F. Hartl. A variable neighborhood search heuristic for periodic routing problems. *European Journal of Operational Research*, 2007. to appear.

[10] K. Hoffman and M. Durbin. The dance of the thirty ton trucks. *Operations Research*, 56(1):3–19, 2008.

[11] G.A.P. Kindervater and M.W.P. Savelsbergh. Vehicle routing: Handling edges exchanges windows. In E. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, USA, 1997.

[12] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[13] J. Kytöjoki, T. Nuortio, O. Bräysy, and M. Gendreau. An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers and Operations Research*, 34(9):2743–2757, 2007.

[14] 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(2):487–499, 2004.

[15] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[16] J.J. Moder, C.R. Phillips, and E.W. Davis. *Project Management with CPM, PERT and Precedence Diagramming*. Van Nostrand Reinhold, New York (NY), USA, 1983.

[17] D. Naso, M. Surico, B. Turchiano, and U. Kaymak. Genetic algorithms for supply-chain scheduling: A case study in the distribution of ready-mixed concrete. *European Journal of Operational Research*, 177(3):2069–2099, 2007.

[18] D. Naso, M. Suricom, and B. Turchiano. Reactive scheduling of a distributed network for the supply of perishable products. *IEEE Transactions on Automation Science and Engineering*, 4(3):407–423, 2007.

[19] ERMCO (European Ready Mixed Concrete Organization). Ready mixed concrete, a natural choice, 2000.

[20] ERMCO (European Ready Mixed Concrete Organization). European ready-mixed concrete industry statistics, year 2004, 2005.

[21] ERMCO (European Ready Mixed Concrete Organization). European ready-mixed concrete industry statistics, year 2005, 2006.

[22] M. Polacek, R. F. Hartl, K.F. Doerner, and M. Reimann. A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics*, 10(6):613–627, 2004.

[23] J. Puchinger and G.R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *IWINAC 2005*, LNCS, pages 41–53, Berlin, 2005. Springer.

[24] V. Schmid, K.F. Doerner, R.F. Hartl, and J.J. Salazar-González. Hybridization of very large neighborhood search for ready-mixed concrete delivery problems. 2008. submitted.

[25] I.D. Tommelein and A. Li. Just-in-time concrete delivery: Mapping alternatives for vertical supply chain integration. In *Proceedings of the Seventh Annual Conference of the International Group for Lean Construction IGLC-7.*, pages 97–108. University of California, Berkeley, 1999.