The Synchronized Dynamic Vehicle Dispatching Problem

Louis-Martin Rousseau,
Michel Gendreau,
Gilles Pesant

CIRRELT - Interuniversitary Research Center on
Enterprise Networks, Logistics and Transportation.
CP 6128 Succ Centre-Ville , Montréal,Canada, H3C 3J7
{louis-martin.rousseau, gilles.pesant,
michel.gendreau}@cirrelt.ca

February 27, 2014

Abstract

This paper solves a particular class of Real Time Vehicle Dispatching Problems where there is need to service some customers with multiple resources subject to synchronization constraints. Real world applications are presented and a flexible heuristic based on Constraint Programming is proposed. In order to facilitate future comparisons, benchmark instances are derived from well-known vehicle routing benchmark instances and a transformation method is detailed. Results on instances containing up to 1000 requests show that solutions are sensitive to the number of synchronization constraints as well as to the requests' arrival rate.

1 Introduction

The area of Vehicle Routing and Scheduling encompasses a rich class of problems with many variations. The focus of this paper is to study a new problem, the Synchronized Dynamic Vehicle Dispatching Problem (SDVDP) where clients occur dynamically and synchronization constraints are present. To our knowledge the SDVDP has not yet been studied in the literature even though it arises in practical contexts such as Dial-a-Ride problems, which sometimes exhibit synchronization constraints when some disabled persons require assistance in order to prepare for transportation. In this case, a special team is sent to the residence of the customer a few minutes before the vehicle to ensure that the transfer is made safely and efficiently. The help can vary from dressing for winter conditions to providing wheelchair assistance and is not usually required

at all times or by all customers. This means that the schedule of the special assistance teams needs to be synchronized with the schedules of the rest of the fleet. This combined problem also occurs in large hospitals where patients are transferred between buildings. The dispatcher who manages the transfers must ensure that the orderlies of both buildings are synchronized with the ambulance.

In a book chapter, Larsen *et al.* [19] retrace the evolution of research on dynamic vehicle routing problems, summarizing research prior to 2000 and focusing on new developments from 2000 on. The authors show how the dynamic case is different from the traditional static vehicle routing problem. They describe the required technological environment and discuss important characteristics of the problem, including the degree of dynamism, elements relevant for the system objective, and evaluation methods for the performance of algorithms. Gendreau and Potvin [12], Qiu and Hsu [25] (in the case of Automated Guided Vehicles), Psaraftis [24], and Ghiani *et al.* [13] have also published surveys describing the different problems and solution approaches. Most efficient algorithms solve this problem with local search techniques ([1], [10], [2]). Ichoua [16] has introduced a taxonomy to further categorize the problems of this class.

Synchronization constraints arise in many real-world contexts such as home care delivery, airline scheduling and ground handling, and forest operations. Examples of such constraints can be found in public service companies that offer different levels of service. The case of the cable companies providing internet services illustrates this problem well. When a customer subscribes to a high speed Internet connection, some providers offer a choice of installation packages: a basic cable modem installation or a more complete software configuration package. The latter installation is usually done by two different technicians and their visits must be synchronized. In fact the company has to ensure that the hardware technician will always precede the software technician and that their visit will be as close to one another as possible. Such problems also appear in distribution systems, when a vehicle cannot pick up a load until another vehicle has first delivered it, or in urban mass transit systems when drivers have to change buses at relief points (see Freling *et al.* [9] and Haase *et al.* [14]).

Bredström and Rönnqvist [5] present a mathematical programming model for a combined vehicle routing and scheduling problem with additional precedence and synchronization constraints. To illustrate the practical significance of the temporal precedence and synchronization constraints, the authors tested their algorithm with the home care staff scheduling problem and discussed the importance of synchronization constraints in the airline industry. In the context of forest operations, the authors discuss the need of coordination between harvesters and forwarders, since forwarding can only be done once the harvesting has been performed. They also describe the requirement to synchronize trucks (without crane) and loaders, since these loaders often serve several stands and move between these in order to load trucks. In the context of airport ground operations, Dohn *et al.* [7] propose a branch-and-price model to a crew rostering application where synchronization constraints are handled implicitly through a branching strategy that iteratively splits time windows. Bredström and Rönnqvist [4] propose a similar Branch and Price algorithm where synchro-

nization constraints are explicitly taken into account in the model and discuss whether they should be included in the master problem or not. The problem of imposing precedence relations, a form of synchronization, was also studied by Malandraki and Dial [21] in the context of the TSP using restricted dynamic programming. The reader is referred to the very thorough survey of [8] for more details.

In this paper we present a Constraint Programming (CP) model for the SDVDP and use this model inside a hybrid metaheuristic to provide solutions to instances with up to 100 customers. Benchmarks are derived from the well-known Solomon instances to show the impact of handling the special request on both the costs and number of serviced customers. A similar methodology was investigated in [3, 2] in the context of a pickup and delivery problem.

The remainder of the paper is organized as follows: Section 2 presents and describes the problem and Section 3, after recalling basic CP notions, presents the model of the instance. Section 4 then details the dynamic heuristic developed to solve the stochastic instance. In Section 5, a transformation is given to produce benchmark instances from known vehicle routing instances and results on those instances are given in Section 6. We conclude in Section 7.

2 The Synchronized Dynamic Vehicle Dispatching Problem

This SDVDP can be described as follows: given a set of customers, a set of vehicles, and a depot, one must find a set of routes starting and ending at the depot and which visits all customers. Each customer has a specific demand, there are capacity constraints on the load that can be carried by a vehicle and each customer must be visited during a specified time window. One can wait in case of early arrival, but late arrival is not permitted. When a customer calls for service the dispatcher must be able to tell the customer in a very short amount of time whether he will be served or not. Some customers ask for service one or more days in advance and some call during the day of operation.

In the synchronized version of this problem, customers can ask for special services. They will be referred to as *special* customers. Such customers will need to be served, in addition to the regular vehicle, by a *special* vehicle. Synchronization requirements such as precedence constraints and restrictions on the time delay between regular and special visits are also specified. These restrictions are given as relative time windows which require the special service of a customer to be provided within a time interval around the regular service time. Capacity constraints are not defined for special vehicles since they usually perform services and not pickups nor deliveries. Furthermore we assume that the service time of both the regular and the special services are equal.

The synchronization constraints make this problem quite hard to solve using traditional local search heuristics. For instance, if the insertion of a special customer delays the route of a special vehicle then all regular visits associated

with that route must also be delayed, independently of the vehicle that served them. But delaying a regular vehicle means delaying also the visits associated with other special customers and so on. This high number of interconnections means that in order to insert a customer, one might have to recompute the visit time of every customer already inserted. As in [2], we have chosen to use CP to model and solve this problem since it not only provides an easy way to express the synchronization constraints but, by representing visit times as domain variables and by propagating the synchronization constraints only when needed, it allows an efficient implementation of those constraints.

The objective of SDVP is to service incoming customers, if possible, at the minimum cost. However, although one would eventually like to service as many customers as possible, we work under the assumption that the service provider is not allowed to deliberately turn down a request in the hope that he will be able to serve more customers later on. When a service requests comes in, the provider must try his best to accommodate it. As a consequence all the following optimization models work under the assumption that minimizing solely the distance driven is both good in terms of reducing cost and creating space for future requests.

3 Constraint Programming Model

Before presenting the model, a brief description of the CP paradigm is given here. This paradigm has been very successful in solving hard combinatorial problems in the field of scheduling, planning, and transportation ([6], [30], [31]). We refer the reader to [27] for more information.

3.1 Constraint Programming

Traditionally a CP model is composed of a set of variables (X), a set of domains (D), and a set of constraints (C) specifying which assignments of values in D to variables in X are allowed. The efficiency of the CP paradigm lies in powerful constraint propagation algorithms which remove from the domain of the variables the values which will generate infeasible solutions. If propagation does not suffice in finding a feasible solution then a branching process is necessary to further narrow the domains; a feasible solution has been found when each domain contains only one value.

Typically at each node of the search tree the following four steps are applied: first an unfixed variable is selected, then a value in its domain is chosen. The selected variable is fixed to the chosen value and constraint propagation occurs. If during propagation the domain of a variable is emptied then the solver has detected an inconsistency in the previously taken decisions and the whole search process backtracks, typically by choosing another value for the variable. When propagation terminates while there are still some unfixed variables, then the solver creates a new search node and goes on with the procedure just detailed. The branching strategy is thus defined by *variable* and *value* selection policies.

This is the simplest and most frequently used branching strategy but more intricate policies are also used. For instance one could split the domain of a variable into two sets of similar size or even use two opposing constraints to define two branching directions.

It is fairly simple to extend this algorithm to solve combinatorial optimization problems, that is to identify the feasible solution which minimizes (or maximizes) a given objective function. Once a feasible solution has been identified, the set C is extended to contain a new constraint specifying that future feasible solutions should have a strictly better cost than the cost of the solution just identified. The solver will thus keep searching for solutions of improving quality until it can prove that there are none (i.e. the last one it found was the optimal solution).

3.2 SDVRP Model

Each *regular* customer is associated to service node $i \in N_r$ and each *special* customer is represented by two nodes: one regular $i \in N_r$ and an associated special node $\hat{\imath} \in N_s$. N is the set of all nodes ($N = N_r \cup N_s$). The depot is copied $2|V|$ times, where V is the set of vehicles, so that each route starts and ends at a different depot. V_r and V_s denote respectively the sets of regular and special vehicles ($V = V_r \cup V_s$) and let I and F be respectively the set of initial and final depots. Note that in the following model (M), regular and special nodes use the same set of variables as they can be differentiated by their indices.

3.2.1 Parameters

D_{ij}	Distance from node i to node j.
T_{ij}	Travel time from node i to node j.
L_i	Load to take at node i.
Q_i	Service time at node i.
A_i, B_i	Bounds on node i's time window.
$R^A_{\hat{\imath}}, R^B_{\hat{\imath}}$	Bounds time window for special service of customer $\hat{\imath}$, which is relative to the time of service at node i.
K	Capacity of the vehicles.

3.2.2 Variables

$s_i \in N_r \cup F$	$\forall i \in N_r \cup I$	Successor of regular node i.
$s_{\hat{\imath}} \in N_s \cup F$	$\forall \hat{\imath} \in N_s \cup I$	Successor of special node $\hat{\imath}$.
$v_i \in V_r$	$\forall i \in N_r$	Vehicle servicing regular node.
$v_{\hat{\imath}} \in V_s$	$\forall \hat{\imath} \in N_s$	Vehicle servicing special node $\hat{\imath}$.
$t_i \in [A_i, B_i]$	$\forall i \in N_r \cup I \cup F$	Time of visit of regular node i.
$t_{\hat{\imath}} \in [A_i - R^A_{\hat{\imath}}, B_i + R^B_{\hat{\imath}}]$	$\forall \hat{\imath} \in N_s$	Time of visit of special node $\hat{\imath}$.
$l_i \in [0, K]$	$\forall i \in N_r \cup I \cup F$	Truck load at regular node i.

3.2.3 Objective

$\min \sum_{i \in N} D_{is_i}$	Minimizing the total distance travelled

Note that in CP it is possible to index an array with a variable through the use of the `element` constraint. Therefore $D_{i_{s_i}}$ represents the distance from i to its successor.

3.2.4 Constraints

$\mathit{AllDifferent}(s_{i:i \in N})$		(1) All successors are different.
$\mathit{NoSubtour}(s_{i:i \in N})$		(2) Subtour elimination constraint.
$s_i = j \Rightarrow l_i + L_i = l_j$	$\forall i \in N_r$	(3) Capacity constraints.
$s_i = j \Rightarrow t_i + Q_i + T_{ij} \leq t_j$	$\forall i \in N$	(4) Time window constraints.
$s_i = j \Rightarrow v_i = v_j$	$\forall i \in N$	(5) Vehicle identification.
$s_i = \hat{\imath} \Leftrightarrow s_{\hat{\imath}} = \hat{\imath}$	$\forall \hat{\imath} \in N_s$	(6) Regular and special node association.
$t_i - R^A_{\hat{\imath}} \leq t_{\hat{\imath}} \leq t_i + R^B_{\hat{\imath}}$	$\forall \hat{\imath} \in N_s$	(7) Synchronization constraint.

The *AllDifferent* (1) constraint is used to enforce on the vector of successor variables s the conservation of flow in the network. The nature of the decision variables already enforces that each node has exactly one successor, but it is also necessary to make sure it has exactly one predecessor. To do so it suffices to ensure that no two nodes have the same successor, which is the role of the *AllDifferent* constraint. This property is obtained by solving a matching problem in a bipartite graph and by maintaining arc consistency in an incremental fashion as described in [26].

The *NoSubtour* constraint (2) (figure 1), which acts on the vector of successor variables s, is taken from the work of Pesant *et al.* [22]. For each chain (sequence of consecutive nodes), the first (β) and last (ε) visits are stored and when two chains are joined together (when a variable is fixed and a new arc is introduced), two actions are taken. First, the information concerning the first and last visits of the new (larger) chain are updated, and then, the value of the first node is removed from the domain of the s variable of the last node.

Constraints sets (3) and (4) enforce that time windows and capacity constraints are respected at regular nodes, while (5) links the vehicle variables v to the successor variable s. We also introduce constraint set (6) to make sure that the regular and the special nodes of a special customer must be both visited or rejected together. We represent the fact that a customer is not serviced by setting the successor variable of its associated node to point on themselves. Constraints (7) are the synchronization constraints which ensure that for each special customer the visit to the special node is coordinated with the visit of the regular node.

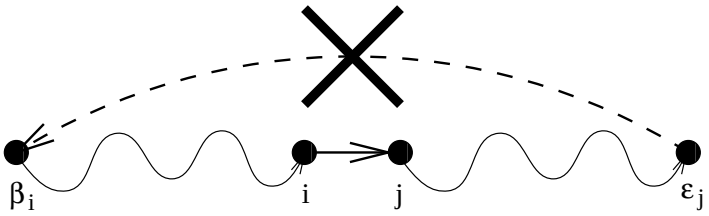

Figure 1: NoSubtour constraint

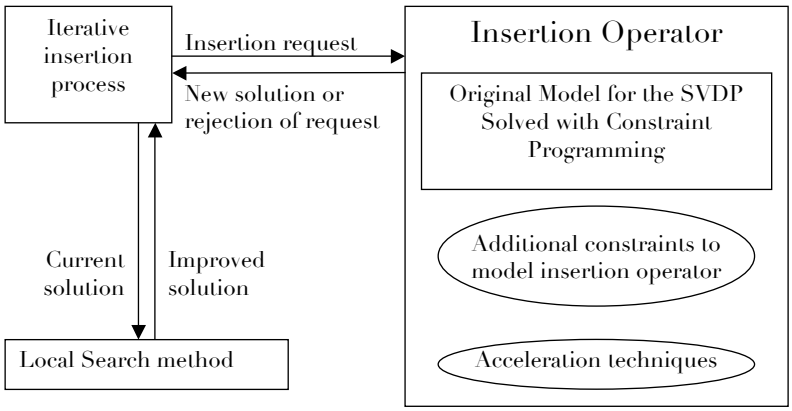

Figure 2: Overall Heuristic

4 Dynamic Solution Heuristic

Building an exact algorithm for this SDVDP is very hard since the dynamic and real time components prevent any solution method from having all the information and decision flexibility at the same time. In the early stages of the dispatching process, the algorithm does not possess the list of all the customers who will require service and thus cannot perform optimal planning. At the end of the process, when most of the information is known, vehicles are already on the road and have served a good number of customers. These decisions cannot be undone and directly impact the ability to service new requesting customers. Moreover, calling customers need to be told whether they will be served or not in a very short amount of time (typically a few seconds), which further limits the possibility of building optimal solutions.

The solution heuristic proposed in this paper relies on the successive insertion of customers as they request service, while local search methods are applied between requests. This kind of procedure is very straightforward and has been applied in numerous real time dispatching algorithms. The first methods to solve real time dispatching problems were proposed in the 1990's by Wilson

and his colleagues ([33], [34], [32]) in the context of a demand-responsive transportation system in Haddonfield, NJ. These algorithms used simple insertion to incorporate each new customer into the current solution. This scheme was later applied ([28], [20]) to disabled person transportation systems when some of the requests were known in advance. The metaheuristics proposed in [23], [11], and [10] were developed for the static version of the problem and adapted to deal with the dynamic context. They essentially perform local search until some event (usually a new request) stops the search and triggers the insertion operator.

The general heuristic is illustrated in Figure 2. When a customer request becomes known, the *Iterative insertion process* asks the *Insertion Operator* whether the new node can be inserted in the current solution. The *Insertion Operator* then builds a CP model of insertion, which can be further strengthened through the use of *Additional Constraint* and *Acceleration Techniques*. After the insertion is performed, or rejected, the *Iterative insertion process* then hands over the control to the *Local Search* that will make use of the idle time, before the next request, to improve the current solution.

4.1 Insertion Procedure

There are many possibilities for insertion strategies: insertion in the first possible position and insertion in the best possible position are two of the most popular ones. Implementing these operators in CP allows easy handling of complex constraints such as synchronization. The rest of this section will detail the implementation of the insertion operators using the model previously presented and give insight into the local search used to improve solution quality.

4.1.1 Constraint-Based Insertion

The insertion operator can be modelled using CP. Starting from a current solution (visiting some of the customers who have previously requested service) and given that s' is a copy of the values of s in the current solution, that the set of previously inserted (special) nodes is defined by P ($\hat{P}$), and that the node(s) to insert is given by the index $k(\hat{k})$, performing the insertion is then only a matter of solving model M with the additional following constraints (yielding M').

$s_k \in P \cup F$		k must precede an inserted node.
$s_i \in \{s'_i, k\}$	$\forall i \in N_r \backslash \{k\} \cup I$	Only simple insertions are permitted.

And the equivalent if k is a special node.

$s_{\hat{k}} \in \hat{P} \cup F$		$\hat{k}$ must precede an inserted special node.
$s_i \in \{s'_{\hat{i}}, \hat{k}\}$	$\forall \hat{i} \in N_s \backslash \{\hat{k}\} \cup I$	Add only if k needs special visit.

When customer k's request (and nodes) becomes known, most vehicles are on the road and some visits have already been performed. Since the previous constraints do not deal with the real time dimension of the SDVDP, it is necessary

to add two additional sets of constraints. The first set enforces that at the current time denoted w, every node that has not already been visited should be visited later than w. The second one forbids insertion after a node when the vehicle serving it has already left[1]. By letting the latest departure time of each node i be denoted by d_i (here $d_i = \min(B_i, R^B_{s_i} - T_{is_i})$), the following update can be performed before the insertion search is started.

$$d_i > w \rightarrow t_i \geq w \quad \forall i \in N$$
$$d_i \leq w \rightarrow s_i = s'_i \quad \forall i \in N$$

Finding a feasible solution to the problem defined by model M' enriched with these additional constraints (yielding M'') gives a *first insertion* solution servicing c k, while solving the problem to optimality yields the *best insertion* of k. It is obvious that the best insertion policy should provide better results since it explores a larger space of solutions, although once combined with a local search procedure (see section 4.2) it is not clear whether this advantage remains. The results section of the paper will compare both insertion strategies. Finally, if no solution to the modified (M'') problem exists then the new node(s) cannot be inserted and the customer request will be rejected.

4.1.2 Acceleration Techniques

The model previously defined can easily tackle all small and medium size instances, ranging from 100 to 400 customers, but it significantly slows down when the number of customers exceeds 400. In order to accelerate the heuristic so that it can solve instances with up to 1000 requests, further filtering techniques are introduced.

The first one is a redundant constraint taken from the work of Pesant *et al.* [22] that allows the detection of inconsistencies in the time windows earlier in the search tree. This constraint, which is checked every time the bounds of one of the t variables is modified, stipulates that :

$$\min(t_i) + Q_i + T_{ij} > \max(t_j) \quad \Rightarrow s_i \neq j \quad \forall\, i, j \in N.$$

One can also note that the bounds on the domains of the resource variables (t and l) of the current solution are valid for the solution obtained after a new customer is inserted. For instance the latest time of visit of each node cannot be extended after an insertion occurs. This means that the bounds obtained in a given solution can be applied directly in all following insertion problems. From this observation another filtering algorithm can be derived. Given that t' and l' are copies of the bounds of the t and l variables of the last solution, the following constraints can be added to the model:

[1]This restriction was lifted in Ichoua [16] which proposes to diverge enroute vehicles in order to serve a new customer.

$$\min(t'_i) \leq t_i \leq \max(t'_i) \quad \forall i \in N$$
$$\min(l'_i) \leq l_i \leq \max(l'_i) \quad \forall i \in N.$$

Since capacity is additive in a very straightforward manner (as opposed to the time dimension which can suffer waiting delays), it can also be checked before the insertion process begins. This yields another set of constraints which accelerate the search process, where k is again given as the index of the node to insert:

$$L_k + \min(l'_j) > K \quad \Rightarrow v_k \neq j \quad \forall j \in F.$$

The variable selection policies traditionally used to solve this kind of problem are based on a *first fail* strategy, which means that the solver tries to branch on the most constrained variable first (usually the one with the smallest domain) in order to detect inconsistencies as early as possible in the search tree. However this strategy is not very effective in the context of the proposed insertion strategy because, after all the insertion constraints have been introduced, the domains of all (but one) s variables contain only two values (the previous successor and the node to be inserted). The only s variable that contains more values is the one associated with the new request. Fixing this variable is equivalent to choosing the insertion position and for this reason s_k is always chosen as the first variable of the branching process. As for values, they are selected in a variable's domain on a most promising basis, meaning the value associated with the minimum cost.

4.2 Local Search

The insertion process is a greedy procedure in the sense that it does not reconsider the relative order of the inserted nodes nor the vehicle they have been assigned to. To overcome this limitation, the dispatching system can make use of the idle time between requests of service to improve the current planning. This improvement phase is achieved with local search, where small modifications are made on the current solution in an attempt to improve its quality.

The local search process is defined using a set of operators and a policy on how to use them (metaheuristic). The set of selected operators are the standard ones used in vehicle routing applications (*2-Opt, Or-Opt, Cross, Exchange* and *Relocate*). These operators are used in conjunction with two different metaheuristics: a simple descent strategy and a more complex Guided Local Search (GLS) [18]. Guided Local Search, like tabu search, is allowed to visit solutions of decreasing quality to avoid local minima. GLS penalizes certain moves which tend to revisit already explored regions of the search space, just like tabu search which forbids those moves. All the operators and metaheuristics are taken from the ILOG Dispatcher library [17].

Since the local search procedure can only be used between customer calls for

service, the amount of time spent improving the solution is not known *a priori*. Therefore the guided local search metaheuristic, which like tabu search can run continually until a new request comes, is more appropriate in a real world situation because it uses the maximum amount of time available for improvement. In a simulation context however, a descent strategy which stops after reaching a local minimum has the advantage of having a performance that is, to a certain point, independent from simulation time. The results section provides a performance comparison for both algorithms.

5 Benchmarks

This section presents benchmark instances developed for the SDVDP which are used to measure the effectiveness of the problem formulation and associated heuristics. The static Vehicle Routing Problem with Time Windows (VRPTW) is probably one of the most benchmarked optimization problems. In 1987 Solomon proposed a set of instances [29] which were to become the reference VRPTW benchmarks.

5.1 Transformation Method

These instances are divided into three sets: the geographical data are randomly generated in instance set R, clustered in instance set C, and a mix of random and clustered structures in instance set RC. The nodes coordinates are identical for all instances within one type (i.e., R, C and RC). The instances differ with respect to the width of the time windows. Some have very tight time windows, while others have time windows which are hardly constraining. These 100 customer instances have later been extended by Homberger and Gehring [15] to larger instances with up to 1000 customers. This set of instances, even if it is not representative of real life instances, facilitates the comparison between methods that solve similar instances.

In the dynamic Vehicle Routing Problem field, most researchers have built instance generators that construct instances according to the probability distributions which best represent the dynamic contexts that their algorithms solved. This yields test instances similar to those that could be encountered in the real world but makes comparisons between algorithms much more difficult.

To facilitate further research on the SDVDP, a set of dynamic benchmark instances is derived from the well-known Solomon instances. The transformation method which renders the static intances dynamic is first presented followed by the variant which creates the synchronized versions.

The Solomon instances describe a set of customers defined by integers, a set of coordinates in the plane, a quantity of product requested, a time window, and a service time. To generate dynamic instances it is only necessary to determine at which point in time each request becomes known to the dispatcher. Some instances may also show different levels of dynamism, meaning that a certain proportion of the requests are known in advance and that a pre-planning phase

is possible before the real-time dispatching process begins. This is namely the case when customers call one or several days prior to their service day. Such instances are generated by selecting a number of customers which constitute a static instance of the VRPTW. This instance can be solved by any algorithm and the best solution found is then used as the starting initial solution for the first customer insertion.

The dynamic instances are generated with the following parameters: customers whose time windows start at time 0 (those who can be served right away) are considered to be static customers. All other visits i are to be requested at time of A_i/α. For instance, α set to 2 means customers call for service at a time equivalent to half of their time window lower bound.

Once this is done, building instances subject to synchronization is just a matter of choosing which customers are going to be special, setting the number of special vehicles and giving values to special visit time windows. In the proposed transformation method, the special customers are distributed evenly across the dispatching horizon. The percentage of customers who will be considered special is given by λ and the number of special vehicles is equal to λ times the number of vehicles usually needed to solve the instance[2]. To select the special customers among those present in the original benchmark files, the following rules are applied:

$$(i \mod (100/\lambda) > 0) \quad \Rightarrow \quad i \in N_r \ \ \forall i \in N$$
$$(i \mod (100/\lambda) = 0) \quad \Rightarrow \quad i \in N_s \ \ \forall i \in N.$$

5.2 Benchmark Instances

In order to simulate the arrival and dispatching process of a normal day of operations using the transformed benchmark instances, it is necessary to generate a mapping between the different time components specified in the test files (like time windows for instance) and the real time during the simulation process. Since the instance sizes varies greatly a fixed simulation time wouldn't yield representative results (i.e. smaller instances would be too easy). We thus need to compute a *per customer* simulation time. This mapping is achieved by defining γ, the average time interval between requests, and using T_{max}, the depot's closing time in the test file. A total runtime R is first computed as the product of γ by the total number of customers ($R = \gamma * |N|$), then a mapping factor $\mu = T_{max}/R$ is defined to link the two time dimensions. Once μ is defined, clock time can be converted to problem time by a simple multiplication.

A set of synchronized instances was created by setting γ to 10 seconds and λ to 10%, meaning one out of 10 customers is considered to be special. The value of $(R^A_{\hat{c}}, R^B_{\hat{c}})$ was set to (10,0) for each special customer $\hat{\imath}$ so that the special visit could start up to 10 units of time before the regular visit.

[2]For the Homberger instances, these numbers were extrapolated from the solutions to the Solomon instances of size 100. For instance C.2.200.01 was said to need twice as many vehicles as C.2.100.01

Size	First Insertion			Best Insertion		
Size	No LS	Desc.	GLS	No LS	Desc.	GLS
100	20.8	11.1	11.4	16.3	10.6	10.6
200	8.5	4.6	4.8	6.8	4.6	4.7
400	10.5	7.4	7.5	8.4	6.7	6.6
600	11.3	10.0	10.0	9.5	8.8	8.7
800	14.1	13.5	13.5	12.7	12.2	12.3
1000	16.6	16.4	16.4	17.6	17.5	17.5

Table 1: Average percentage of rejected customers for the SDVDP, with $\alpha = 2$, $\lambda = 10\%$ and $\gamma = 10$ sec.

6 Experimental Results

This section describes the performance of the proposed heuristic for the SDVP on the previously described benchmark. Table 1 reports the proportion of rejected customers against instance size using the proposed insertion operators and local search. There are 56 instances of size 100 and 60 instances of size 200, 400, 600, 800, and 1000 for a total of 356 instances containing 185 600 requests.

Table 1 shows the rejection ratio yielded by the different heuristics when applied to synchronized instances. The main factor of performance for the smaller instances seems to be local search, whatever form it takes, since simple local search is as effective as Guided Local Search. For the larger instances however, *Best Insertion* outperforms local search. The combination of both techniques yields even better results except for the instances of size 1000, where *Best Insertion* takes all the time available to evaluate a request and no local search can be applied. In the following tables the overall best combination (and the one most suited to real life cases) of GLS and *Best Insertion* will be used to measure the impact of the γ and λ parameters, that is the rate of arrival of new requests and ratio of special customers.

We first note the significant difference of rejection ratios between the 100 series and the rest of the instances. This discrepancy is probably due to the fact that the instances of size 100 were generated by Solomon [29] while the larger instances were later proposed by Homberger [15] and thus present a different structure (although [15] indicates they were built in a similar fashion). For the instances of size 200 to 1000, the rejection ratio increases with the size of the instances. We believe this can be explained by the fact that when instances grow in size the difficulty of performing successive insertions increases and the time available for insertion and local search becomes insufficient.

A key factor that influences the rejection ratio of customers is the average time between requests. Table 2 clearly indicates that when more time is available to perform insertion and local search, more customers, both special and regular, can be served. It seems however that for smaller instances (100, 200) a time interval of 20 seconds is sufficient and that increasing the value of γ does

Size	Time interval between requests							
Size	$\gamma = 5$		$\gamma = 10$		$\gamma = 20$		$\gamma = 40$	
	Reg.	**Spec.**	**Reg.**	**Spec.**	**Reg.**	**Spec.**	**Reg**	**Spec.**
100	9.4	16.7	10.0	15.7	9.0	16.6	9.0	16.2
200	3.6	17.6	3.3	18.2	3.2	17.2	3.2	16.9
400	5.9	23.5	4.9	22.4	4.1	21.0	4.0	20.3
600	7.7	27.0	6.6	27.1	5.6	25.5	4.9	24.1
800	12.5	26.6	11.0	24.3	9.5	23.9	8.5	22.5
1000	16.4	29.1	16.0	30.5	15.0	30.9	12.6	28.6

Table 2: Percentage of rejected customers for the SDVDP with $\alpha = 2$, $\lambda = 10\%$

Size	Percentage of special customers							
Size	$\lambda = 5\%$		$\lambda = 10\%$		$\lambda = 25\%$		$\lambda = 50\%$	
	Reg.	**Spec.**	**Reg.**	**Spec.**	**Reg.**	**Spec.**	**Reg**	**Spec.**
100	9.9	14.0	10.0	15.7	9.0	28.7	10.6	22.6
200	3.3	14.7	3.2	18.2	3.3	11.6	3.2	11.9
400	5.1	24.5	4.9	22.4	5.1	18.5	5.4	15.4
600	6.6	31.5	6.6	27.1	6.8	21.6	6.9	18.1
800	10.5	27.3	11.0	24.3	10.9	22.5	11.6	20.0
1000	15.4	33.5	16.0	30.5	18.1	29.1	23.1	23.9

Table 3: Percentage of rejected customers for the SDVDP with $\alpha = 2$, $\gamma = 10$

not really improve solutions. Instances of size 400 and above would however probably benefit from increased time intervals.

Table 3 reports the rejection ratio according to the percentage of special customers. By looking at series 400 to 1000 we note that, although the percentages for regular customers are quite stable, the percentage of rejected special customers decreases when λ increases. This is explained by the flexibility gained with the increased number of special vehicles. Thus, when the number of special customers rises, more of these customers are rejected in absolute value, but it becomes relatively easier to insert them.

The conflicting results for 100 and 200 series are caused by the discrete nature of the number of special vehicles. In these instances there are too few regular vehicles to generate the proper ratio of special vehicles. This means that each instance either has too many or too few special vehicles in proportion to regular ones.

7 Conclusion

We proposed a heuristic based on CP to tackle a dynamic Vehicle Routing Problem in the presence of synchronization constraints.These synchronization

constraints could be applied in the context of any dynamic fleet assignment problem.

Constraint-based insertion operators were derived from the original model for the Synchronized Dynamic Vehicle Dispatching Problem and were used to define a successive insertion procedure. Local search was also applied between requests to improve the current solution and help future insertions. Results were given on a set of constructed benchmark instances to demonstrate the relative performance of the insertion operators and local search proposed, as well as the sensitivity to the rate of arrival of the requests and the ratio of special customers.

The problem was solved using CP for mainly two reasons. Firstly the modelling power of this paradigm, which greatly eased the modelling phase, allows for a complete separation between the model and the search procedure. This feature makes it simple to add new constraints without having to modify the insertion operator or the local search method. Secondly but most importantly, the domain variable representation and the propagation techniques of CP eased the handling of synchronization constraints.

References

[1] A. Attanasio, J.-F. Cordeau, G Ghiani, and G. Laporte. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3):377–387, 2004.

[2] G. Berbeglia, J.-F. Cordeau, and G. Laporte. A hybrid tabu search and constraint programming algorithm for the dynamic dial-a-ride problem. *INFORMS Journal on Computing*, 24(3):343–355, 2012.

[3] G. Berbeglia, G. Pesant, and L.-M. Rousseau. Checking the feasibility of dial-a-ride instances using constraint programming. *Transportation Science*, 45(3):399–412, 2011.

[4] D. Bredström and M. Rönnqvist. A branch and price algorithm for the combined vehicle routing and scheduling problem with synchronization constraints. Technical Report Discussion Paper No 2007/07, Dept. of Finance & Management Science, NHH, 2007.

[5] D. Bredström and M. Rönnqvist. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operational Research*, 191(1):19–31, 2008.

[6] M. Dincbas, P. Van Hentenryck, and H. Simonis. Solving Large Combinatorial Problems in Logic Programming. *Journal of Logic Programming*, 8:75–93, 1990.

[7] A. Dohn, E. Kolind, and J. Clausen. The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach. *Computers & Operations Research*, 36(4):1145–1157, 2009.

[8] M. Drexl. Synchronization in Vehicle Routing - A Survey of VRPs with Multiple Synchronization Constraints. *Transportation Science*, 46(3):297–316, 2012.

[9] R. Freling, D. Huisman, and A. P. M. Wagelmans. Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling*, 6(1):63–85, 2003.

[10] M. Gendreau, F. Guertin, J.-Y. Potvin, and R. Séguin. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research, Part C*, 14(3):157–174, 2006.

[11] M. Gendreau, F. Guertin, J.-Y. Potvin, and E. Taillard. Tabu Search for Real-Time Vehicle Routing and Dispatching. *Transportation Science*, 4(33):381–390, 1999.

[12] M. Gendreau and J.-Y. Potvin. Dynamic Vehicle Routing and Dispatching. *Fleet Management and Logistics*, pages 115–126, 1998.

[13] G. Ghiani, F. Guerriero, G. Laporte, and R. Musmanno. Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies. *European Journal of Operational Research*, 151(1):1–11, 2003.

[14] K. Haase, G. Desaulniers, and J. Desrosiers. Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Science*, 35(3):286–303, 2001.

[15] J. Homberger and H. Gehring. Two Evolutionary Metaheuristics for the Vehicle Routing Problem with Time Windows. *INFOR*, 37:297–318, 1999.

[16] S. Ichoua. *Problème de gestion de flottes de véhicules en temps réel.* PhD thesis, Université de Montréal, 2001.

[17] ILOG S.A. *ILOG DISPATCHER*, 2002.

[18] P. Kilby, P. Prosser, and P. Shaw. Guided local search for the vehicle routing problem with time windows. In Stefan Vo, Silvano Martello, IbrahimH. Osman, and Catherine Roucairol, editors, *Meta-Heuristics*, pages 473–486. Springer US, 1999.

[19] A. Larsen, O.B.G. Madsen, and M.M. Solomon. Recent developments in dynamic vehicle routing systems. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces*, pages 199–218. Springer US, 2008.

[20] O.B.G. Madsen, H.F. Ravn, and J.M. Rygaard. A Heuristic Algorihtm for a Dial-a-Ride Problem with Time Windows, Multiple Capacities and Mulptiple Objectives. *Annals of Operations Research*, 61:213–226, 1995.

[21] C. Malandraki and R. B. Dial. A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *European Journal of Operational Research*, 90(1):45–55, 1996.

[22] G. Pesant, M. Gendreau, J.-Y. Potvin, and J.-M. Rousseau. An Exact Constraint Logic Programming Algorithm for the Traveling Salesman Problem with Time Windows. *Transportation Science*, 32:12–29, 1998.

[23] H.N. Psaraftis. Dynamic Vehicle Routing Problems. In B.L. Golden and A.A. Assad, editors, *Vehicle Routing : Methods and Studies*, pages 223–248. North-Holland, 1988.

[24] H.N. Psaraftis. Dynamic vehicle routing: Status and prospects. *Annals of Operations Research*, 61(1):143–164, 1995.

[25] L. Qiu and W. Hsu. Scheduling and Routing Algorithms for AGVs: a Survey. Technical Report CAIS-TR-99-26, Centre for Advanced Information Systems, School of Applied Science, Nanyang Technological University, Singapore, 1999.

[26] J.-C. Régin. A filtering algorithm for constraints of difference in csps. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, AAAI '94, pages 362–367. American Association for Artificial Intelligence, 1994.

[27] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming* . Elsevier, 2006.

[28] S. Roy, J.-M. Rousseau, G. Lapalme, and J. Ferland. Routing and Scheduling of Transportation Services for the Disabled: Summary Report. Technical Report CRT-473A, Centre de Recherche sur les Transports, Université de Montréal, 1984.

[29] M. M. Solomon. Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints. *Operations Research*, 35:254–265, 1987.

[30] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge, Mass., 1989.

[31] M. Wallace. Practical Applications of Constraint Programming. *Constraints*, 1:139–168, 1996.

[32] N.H.M. Wilson and N.H. Colvin. Computer Control of Rochester Dial-a-Ride System. Technical Report R-77-30, Departement of Civil Engineering, Massachusetts Institute of Technology, 1977.

[33] N.H.M. Wilson, J.M. Sussman, H.K. Wong, and B.T. Higonnet. Scheduling Algorithms for Dial-a-Ride Systems. Technical Report USL TR-70-13, Urban Sustems Laboratory, Massachusetts Institute of Technology, 1971.

[34] N.H.M. Wilson and H. Weissberg. Adavance Dial-a-Ride Algorithms Research Project: Final Report. Technical Report TR-76-20, Departement of Civil Engineering, Massachusetts Institute of Technology, 1976.