TRANSPORTATION SCIENCE
Vol. 40, No. 1, February 2006, pp. 74–88
ISSN 0041-1655 | EISSN 1526-5447 | 06 | 4001 | 0074

DOI 10.1287/trsc.1050.0133

Dynamic Column Generation for Dynamic Vehicle Routing with Time Windows

Zhi-Long Chen
Robert H. Smith School of Business, University of Maryland, College Park, Maryland 20742-1815, zchen@rhsmith.umd.edu

Hang Xu
Oracle Corporation, 25 First Street, Suite 303, Cambridge, Massachusetts 02141, henry.xu@oracle.com

We consider a dynamic vehicle routing problem with hard time windows, in which a set of customer orders arrives randomly over time to be picked up within their time windows. The dispatcher does not have any deterministic or probabilistic information on the location and size of a customer order until it arrives. The objective is to minimize the sum of the total distance of the routes used to cover all the orders. We propose a column-generation-based dynamic approach for the problem. The approach generates single-vehicle trips (i.e., columns) over time in a real-time fashion by utilizing existing columns, and solves at each decision epoch a set-partitioning-type formulation of the static problem consisting of the columns generated up to this time point. We evaluate the performance of our approach by comparing it to an insertion-based heuristic and an approach similar to ours, but without computational time limit for handling the static problem at each decision epoch. Computational results on various test problems generalized from a set of static benchmark problems in the literature show that our approach outperforms the insertion-based heuristic on most test problems.

Key words: dynamic vehicle routing; time windows; column generation; heuristics
History: Received: June 2003; revision received: June 2005; accepted: August 2005.

1. Introduction

A tremendous amount of research on vehicle routing problems has been published in the past four decades. The vast majority of the vehicle routing literature is dedicated to deterministic and static models where all problem parameters are assumed to be known in advance and decisions are made only once at the beginning of the planning horizon. However, distribution operations in practice often involve uncertainties with respect to locations and sizes of customer orders, vehicle travel times, etc. In practical vehicle routing applications arising in courier mail services, residential utility repair services, and distribution of heating oils or liquid gas to private households, typically, at the beginning of a planning horizon (e.g., at 9:00 A.M. in a day), only a portion (e.g., 50%) of customer orders to be covered in the planning horizon are known. The rest will arrive randomly over time. The location and size of an order will become known only by the time when this order arrives. This implies that optimal vehicle routes planned at the beginning of the planning horizon will no longer be optimal or even feasible some time later, and have to be modified over time by incorporating up-to-date information. Recent advances in communication and information technologies make it easier and more affordable to update data in real time. This enables dispatchers to make real-time routing decisions.

Most results on dynamic vehicle routing models were published in the last 15 years. Surveys of existing results can be found in Powell (1988), Powell, Jaillet, and Odoni (1995), Psaraftis (1995), Bertsimas and Simchi-Levi (1996), and Gendreau and Potvin (1998). Most recent results can be found in the papers reviewed below and the references cited there. Two major classes of solution approaches have been reported in the literature. They differ mainly in the way with which the dynamism of the underlying problem is dealt. One class of methods, hereafter called *local approaches*, plan and replan routes solely based on known information without looking into the uncertain future. At each decision epoch, a static problem consisting of known orders up to this point in time that have not been covered is solved. These methods do not need any advance information about future events and can be used for situations where future orders are difficult to predict. The other class of methods, hereafter called *look-ahead approaches*, tries to incorporate probabilistic features of future events or forecasted future information into the static problem at each decision epoch. These approaches require advance information about future events and can be used for situations where at least some probabilistic information about future orders is known in advance.

Recent results that have studied look-ahead approaches for dynamic routing problems can be found

in, among others, Powell (1996), Powell and Carvalho (1998), Godfrey and Powell (2002a, b), Bent and Van Hentenryck (2004), and Larsen, Madsen, and Solomon (2004). Because the algorithm we will develop in this paper is a local approach, we will not review these papers.

Recent results on local approaches for dynamic routing problems include, among others, Regan, Mahmassani, and Jaillet (1996, 1998), Shieh and May (1998), Gendreau et al. (1999), Ichoua, Gendreau, and Potvin (2000), Mahmassani, Kim, and Jaillet (2000), Larsen, Madsen, and Solomon (2002), and Yang, Jaillet, and Mahmassani (1999, 2004). There are three types of algorithms reported in these papers for solving the static problem at each decision epoch: (i) sophisticated optimization-based algorithms, (ii) simple rule-based or local-search-based algorithms, (iii) hybrids of optimization-based and rule-based algorithms. Regan, Mahmassani, and Jaillet (1996, 1998), Mahmassani, Kim, and Jaillet (2000), and Yang, Jaillet, and Mahmassani (1999, 2004) all consider the problem of dynamically assigning vehicles to pickup-and-delivery loads in which each load has a delivery deadline such that a penalty is incurred if a load is delivered after its deadline. The objective is to cover a subset of the loads so as to minimize the sum of total empty distance traveled by vehicles, loss of revenue for the uncovered loads, and penalty for delivery lateness. These papers differ mainly in the approaches they use for handling the underlying static problems: Regan, Mahmassani, and Jaillet (1996, 1998) propose simple rule-based heuristics; Mahmassani, Kim, and Jaillet (2000) propose a hybrid approach; Yang, Jaillet, and Mahmassani (1999) propose an optimization-based algorithm; Yang, Jaillet, and Mahmassani (2004) propose both local search and optimization-based algorithms. Shieh and May (1998) study a dynamic version of the static vehicle routing problem with hard time windows and propose an insertion-based heuristic for the underlying static problems, in which new orders are inserted to existing routes that are then improved by local search. Gendreau et al. (1999) and Ichoua, Gendreau, and Potvin (2000) address a dynamic vehicle routing problem with soft time windows, which is similar to the problem considered by Shieh and May except that the time window of each order is not strict and can be violated at some cost. They propose tabu search-based heuristics for the static problems involved. Larsen, Madsen, and Solomon (2002) consider a dynamic traveling repairman problem where a repairman needs to cover all customer requests, some of which arrive dynamically over time, at a minimum total cost. They compare various rule-based heuristics for problems with varying degrees of dynamism.

Because a local approach does not consider the entire problem at each decision epoch, using an optimization algorithm to solve the underlying static problems to optimality does not necessarily generate a better overall solution for the entire problem than the case in which a simple heuristic is used instead. However, the computational results obtained by Yang, Jaillet, and Mahmassani (2004) on a dynamic vehicle assignment problem show that using optimal approaches for the static problems does generate better overall solutions for their problems than using heuristics. However, optimization-based algorithms have an obvious drawback: They may take a much longer time to respond to new information than simple heuristics. It is clearly a challenge to find a fast and yet robust optimization-based algorithm that can generate a near-optimal solution to the static problem quickly enough so that a sufficient number of static problems can be solved over time to guarantee that new information is responded to in a timely manner in the decision process. We attempt to design such an approach in this paper.

In this paper, we consider a dynamic version of the static vehicle routing problem with hard time windows. Customer orders for pickup service are called in over time in a given planing horizon. Each order is associated with a time window within which the order must be picked up. The location, size, and time window of an order become known only after the order arrives. A sufficient number of homogeneous vehicles for covering customer orders are located at a single depot at the beginning of the planning horizon. The problem is to dispatch vehicles over time to cover all the orders that arrive within the planning horizon so as to minimize the sum of the total distance of the dispatched vehicles. The problem we consider is similar to the ones studied by Shieh and May (1998), Gendreau et al. (1999), and Ichoua, Gendreau, and Potvin (2000) with the following differences. In our problem we require that every order must be covered, whereas there is no such requirement in the problem of Shieh and May; and in our problem the time windows cannot be violated, whereas they can be violated with some penalty in the problems of Gendreau et al. and Ichoua, Gendreau, and Potvin. Because the time window of an order is unknown until the order arrives, it is possible in general that no vehicle (either currently at the depot, or en route to an order) can cover this order within its time window. However, we assume that the arrival time of an order is sufficiently early that it can always be covered by a vehicle from the depot dispatched at the next dispatching time (called implementation epoch later) immediately after the order arrival. The reason we make such an assumption and require that each order must be covered is mainly for ease of evaluation of our solution approach, as will become clear later.

We propose a column-generation-based dynamic approach for the problem. The approach attempts to

solve a static problem at each decision epoch containing all the known orders that have not been picked up so far. No information on future orders is incorporated in this static problem. Therefore, our approach is a local approach by the terminology defined earlier. The approach dynamically generates columns (i.e., single-vehicle trips) over time by alternately solving a linear program and applying a fast local-search-based heuristic. The static problem at each decision epoch is approximated by a set-partitioning-type formulation consisting of the columns generated during the time interval between the last and the current decision epochs. The formulation is solved directly by the MIP solver of CPLEX, and the solution is implemented up to a time point that is a prespecified time (used for computation) later than the next decision epoch.

It is well known that column-generation-based approaches work well for many different classes of static combinatorial optimization problems including the static vehicle routing problem with time windows (e.g., Desrochers, Desrosiers, and Solomon 1992). Our goal in this paper is to extend the static column generation methodology to the dynamic environment and investigate how well it works in that context. Our main idea is to make column generation dynamic in a natural and computationally efficient way. As new orders arrive and old orders are completed over time, new columns (i.e., single-vehicle trips) that reflect the up-to-date information are generated dynamically using fast heuristics by modifying existing columns generated earlier—e.g., removing orders already covered from an existing column, inserting new orders into an existing column, etc. Because the approach generates new columns by utilizing existing columns, it is capable of generating a large number of new columns within a very short period of time. At each decision epoch, a restricted set-partitioning-type formulation of the static problem at this point in time is readily available by pulling together all the columns generated so far. Due to its special structure, a set-partitioning-type formulation with a reasonably large number of columns and rows (e.g., up to 20,000 columns and 100 rows) can normally be solved very quickly directly by CPLEX.

We evaluate our approach against three approaches: (i) a global static approach that assumes that all the information about orders is known in advance and solves the static problem over the entire planning horizon optimally or near optimally; (ii) an approach that uses an insertion-based heuristic for the underlying static problems, which is adapted from the insertion heuristic proposed by Solomon (1987) for the static vehicle routing problem with time windows; (iii) an approach that is similar to ours but is allowed to take an unlimited amount of computational time at each decision epoch such that a much larger number of columns can be considered (if necessary) when solving the underlying static problem at that point in time. We note that an insertion-based heuristic is used as a benchmark mainly because similar approaches have been used in the literature either as a main algorithm (Shieh and May 1998) or as a benchmark for evaluating more sophisticated approaches (Bent and Van Hentenryck 2004). Furthermore, insertion-based heuristics are easy to implement, and hence are often used in practical routing systems (Fu and Teply 1999; Shen et al. 1995). Our computational experiment on various problem instances created from a well-known set of benchmark static problems shows that (i) our approach outperforms the approach that uses an insertion-based heuristic for static problems in most test instances, and (ii) our approach overall is not as good as the approach that is allowed an unlimited amount of computational time for solving static problems; however, the gap between the two approaches is small.

This paper is organized as follows. In §2, we give the formal description of the problem we study. In §3, a dynamic column-generation-based approach for the problem is proposed. In §4, two approaches used for comparison are described. Computational results are reported in §5. Finally, the paper is concluded in §6.

2. Problem Description

We are given a transportation network in a Euclidean plane such that the travel distance between any two points in the network is the straight-line distance between them. A sufficient number of homogeneous vehicles with a limited capacity and a constant speed are located at a single depot with coordinates $(0, 0)$ in the Euclidean plane at time 0, the beginning of the planning horizon $[0, H]$. Customers call in requesting on-site pickup over time. The dispatcher does not have any advance knowledge (either deterministic or probabilistic) of future customer orders, and the location, size, and time window of an order become known with certainty from the moment in which it is called. Each order i can be represented by seven parameters associated with it, denoted as the *parameter vector* $P_i = (ct_i, x_i, y_i, w_i, a_i, b_i, st_i)$, where

ct_i = the call-in time of the order, $0 \le ct_i < H$, before which the dispatcher does not know the existence of the order.

(x_i, y_i) = the coordinates of the location of the order in the Euclidean plane. The order must be picked up at this location and taken to the depot.

w_i = the weight of the order. Total weight of the orders carried by a vehicle cannot exceed the vehicle capacity.

$[a_i, b_i]$ = the time window of the order, $ct_i < a_i < b_i < H$. The time window cannot be violated, i.e.,

order i must be picked up within this time window. A vehicle may have to wait if it intends to pick up order i but comes to the site before a_i.

st_i = the service time (time for order pickup) of the order.

At the beginning of the horizon, i.e., time 0, there is already a set of n_0 initial orders, denoted as $N_0 = \{1, 2, \ldots, n_0\}$, in the system. They were left over from the previous planning horizon or called in between two consecutive planning horizons (e.g., last night or early this morning). The depot has a hard time window equal to the planning horizon $[0, H]$ such that a vehicle, once dispatched, must return to the depot within this time window. The problem is to dispatch vehicles over the planning horizon to cover all the orders that are already in the system at the very beginning and the orders that are called in randomly during the planning horizon so as to minimize the total distance traveled by the vehicles dispatched.

The static version of this problem, commonly referred as VRPTW (vehicle routing with time windows), where all the orders are known at the beginning of the planning horizon, has been widely studied in the literature. Many heuristics and optimization-based algorithms have been proposed for this problem. It is known that VRPTW problems with wider time windows are in general more difficult to solve than those with narrower time windows. Most recent optimization algorithms for this problem include a column-generation-based approach enhanced by strong valid inequalities proposed by Kohl et al. (1999), and a branch-and-cut algorithm by Bard, Kontoravdis, and Yu (2002). Those algorithms can find optimal solutions to problems with up to 100 orders and relatively narrow time windows. VRPTW problems with a larger size and relatively wide time windows are still considered computationally challenging. On the other hand, VRPTW is one of the most basic vehicle routing problems, and numerous practical routing models are generalizations of this problem.

3. A Dynamic Column Generation Approach

In this section, we develop a dynamic column generation (DYCOL) approach for the problem described in §2. Because many other dynamic routing problems are generalizations of the problem we consider here, the framework of DYCOL is applicable to other dynamic routing problems as well. Our approach is rolling horizon based and solves a series of static problems over time. We introduce K *decision epochs* for the problem, $t_1, t_2, \ldots, t_K$ with $t_1 = 0 < t_2 < \cdots < t_K < H$. They are evenly spaced in time, i.e., $t_i = (i-1)\Delta$ for $i = 1, \ldots, K$, where $\Delta > 0$ is the time between two consecutive epochs that can be either prespecified or determined by experiment. At each decision epoch t_i, we solve a static problem consisting of all the information known up to this point in time. We allow a computation time τ for solving the problem. By time $t_i + \tau$, a solution to the problem has been obtained for implementation. The solution is implemented over the time interval $[t_i + \tau, t_{i+1} + \tau]$. We call time $t_i + \tau$ the *implementation epoch* of the solution obtained at the decision epoch t_i. Figure 1(a) illustrates the dynamism of our approach. The details of the static problems involved and how these problems are solved are described in the following subsections.

3.1. The Static Problem at Each Decision Epoch

The static problem at each decision epoch t_i is defined over the horizon $[t_i + \tau, H]$. The orders considered in this problem are the orders that have arrived before or at time t_i, but have not been covered by the next implementation epoch, time $t_i + \tau$. Whether an order will have been covered by time $t_i + \tau$ is determined by the solutions implemented at earlier implementation epochs. Although the solution of this static problem will be implemented at time $t_i + \tau$, the orders that arrive during the period $(t_i, t_i + \tau]$ are not considered in this problem because all the input to this problem has to be known at time t_i when we start to solve this problem. The vehicles involved in the static problem of the decision epoch t_i consist of two sources: (i) the ones that are dispatched earlier but have not returned to the depot by time $t_i + \tau$; and (ii) the ones that are at the depot. There is an infinite number of vehicles from source (ii), and a finite number of vehicles from source (i). Note that the initial status of the vehicles from source (i) is their status at time $t_i + \tau$, given that the horizon of this problem starts at time $t_i + \tau$.

In the following, we specify precisely the set of orders, denoted as N_i, and set of vehicles, denoted as V_i, involved in the static problem at each decision epoch t_i, for $i = 1, 2, \ldots, K$. The static problem at the first decision epoch $t_1 = 0$ consists of the order set $N_1 = N_0$ and the vehicle set V_1 with an infinite number of vehicles located at the depot. Suppose that we have solved the static problem of decision epoch t_{i-1} and obtained a solution S_{i-1} for some $i \in \{2, \ldots, K-1\}$. To implement the solution S_{i-1} at the implementation epoch $t_{i-1} + \tau$, a set of vehicles from V_{i-1} are dispatched to cover the orders in N_{i-1} according to the solution S_{i-1}. By the next implementation epoch $t_i + \tau$, some of the vehicles dispatched may have not completed their planned trips (i.e., have not returned to the depot) and still have some capacity remaining. Denote the set of these vehicles by U_{i-1}. Clearly $U_{i-1} \subset V_{i-1}$ and U_{i-1} is a finite set. Based on the solution S_{i-1}, the exact location and remaining capacity of each of the vehicles in U_{i-1} at time $t_i + \tau$ is known

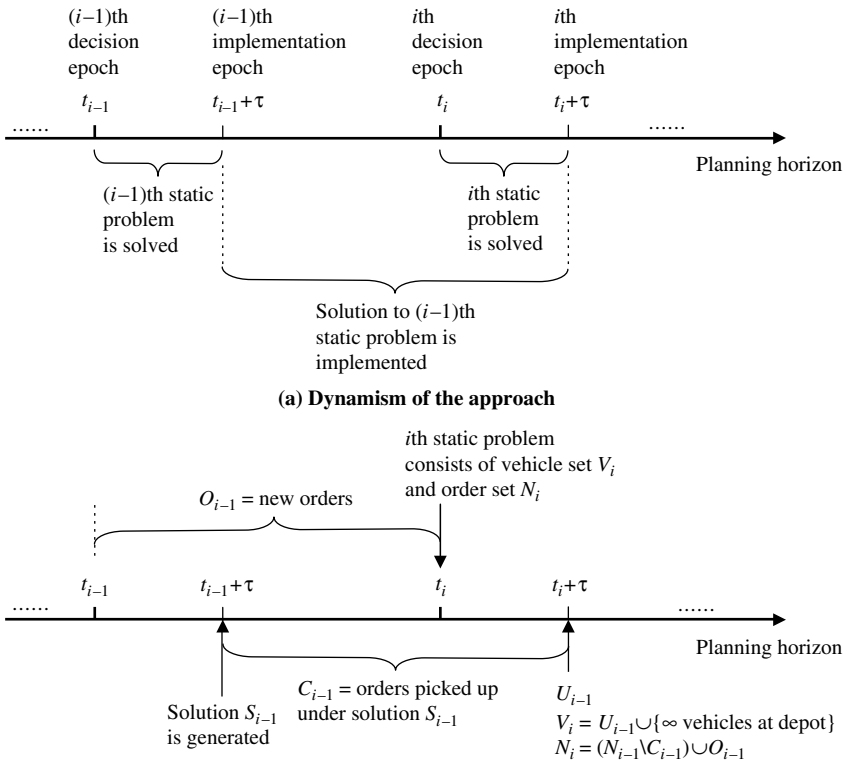

Figure 1 An Illustration of the Structure of the Solution Approach DYCOL

immediately after S_{i-1} is obtained and implemented at time $t_{i-1}+\tau$. In the static problem of the next decision epoch t_i, all the vehicles in U_{i-1} and an infinite number of vehicles located at the depot can be used. Thus, the set of vehicles involved in the static problem of the decision epoch t_i is

$$V_i = U_{i-1} \cup \{\text{infinitely many vehicles located at the depot}\}.$$

Let C_{i-1} ($C_{i-1} \subset N_{i-1}$) denote the set of the orders picked up during the interval $[t_{i-1}+\tau, t_i+\tau]$ under the solution S_{i-1}, and O_{i-1} denote the set of orders that arrive in the interval $[t_{i-1}, t_i]$. Then the set of orders to be considered in the static problem of the decision epoch t_i is

$$N_i = (N_{i-1} \setminus \{C_{i-1}\}) \cup O_{i-1}.$$

Figure 1(b) illustrates V_i and N_i that form the static problem of the decision epoch t_i.

For ease of presentation later, we classify the vehicles in V_i into $q_i = 1 + |U_{i-1}|$ different types based on their location and remaining capacity at time $t_i+\tau$, such that each vehicle in U_{i-1} represents a different type and all the vehicles located at the depot represent a single type. Let $Q_i = \{1, 2, \ldots, q_i\}$ represent the set of the vehicle types associated with V_i, where the first $q_i - 1$ types $1, 2, \ldots, q_i - 1$ correspond to the $q_i - 1$ vehicles in U_{i-1}, and the last type q_i represents the vehicles located at the depot.

Given the set of orders N_i to be covered and the set of vehicles V_i available at time $t_i+\tau$, the static problem of decision epoch t_i is to dispatch a subset of the given vehicles in V_i to cover all the orders in N_i so as to minimize the sum of the total distance traveled by the dispatched vehicles over the horizon $[t_i+\tau, H]$. This problem does not take into account potential impact of its solution on future orders that arrive after time t_i. When solving this problem, the vehicles in U_{i-1} can be diverted to cover orders other than the ones planned under the solution S_{i-1} even if the vehicle is in the course of traveling to somewhere for picking up an order under the solution of S_{i-1}. However, a vehicle that is at the location of an order at time $t_i+\tau$ picking up the order has to complete picking up the order first before it can be diverted to somewhere else other than the ones planned in the solution S_{i-1}. The solution S_{i-1} is implemented up to time $t_i+\tau$, and any orders scheduled under this solution beyond this time are rescheduled, along with newly called-in orders in the problem.

3.2. Getting a Solution to the Static Problem at Each Decision Epoch

The static problem of each decision epoch t_i is a static VRPTW, which is known to be NP-hard, and hence

it would be difficult to find an optimal solution to this problem in a short amount of time. Therefore, we seek to find a near-optimal solution to the problem quickly. To this end, we consider a restricted version of the problem that is computationally more tractable.

During the time interval $[t_{i-1} + \tau, t_i]$, we use a column-generation-based approach (described in §3.3) to dynamically generate a set of feasible single-vehicle trips, denoted as Γ_j, for each vehicle type $j \in Q_i$ covering a subset of orders in N_i. Spatially, these trips originate from the initial locations of the respective vehicles in V_i and end at the depot. Temporally, these trips start at time $t_i + \tau$ and end when the vehicles return to the depot. Define the following parameters f_u and δ_{lu}, and binary variables x_u, for $u \in \Gamma_j$ and $l \in N_i$:

$f_u =$ the total distance of a trip u,
$\delta_{lu} = 1$ if order $l \in N_i$ is covered by a trip u,
$x_u = 1$ if a trip $u \in \Gamma_j$ is used, and 0 otherwise.

Then the following set-partitioning-type formulation is a restricted version of the static problem at the decision epoch t_i.

$$[\mathrm{SP}_i] \quad \text{Minimize} \quad \sum_{j \in Q_i} \sum_{u \in \Gamma_j} f_u x_u \tag{1}$$

$$\text{s.t.} \quad \sum_{j \in Q_i} \sum_{u \in \Gamma_j} \delta_{lu} x_u = 1, \quad \text{for } l \in N_i \tag{2}$$

$$\sum_{u \in \Gamma_j} x_u \le 1, \quad \text{for } j = 1, 2, \ldots, q_i - 1 \tag{3}$$

$$x_u \in \{0, 1\}, \quad \text{for } u \in \bigcup_{j \in Q_i} \Gamma_j \tag{4}$$

This formulation seeks to find a subset of single-vehicle trips of the given set $\cup_{j \in Q_i} \Gamma_j$ so that the total distance of the selected trips is minimum (represented by objective function (1)), each order in N_i is covered (constraint (2)), and each vehicle in U_{i-1} is assigned to at most one trip (constraint (3)). We note that if Γ_j contains *all* feasible trips for each vehicle type $j \in Q_i$, then this formulation represents exactly the static problem of the decision epoch t_i. However, our solution approach described in §3.3 does not generate or evaluate all feasible trips for a vehicle type, and the set Γ_j we use is only a small subset of all the feasible trips. This is why we refer to this formulation as a *restricted version* of the underlying problem.

At each decision epoch t_i, we solve the binary integer program $[\mathrm{SP}_i]$ by calling the MIP solver of CPLEX. Due to the special structure of this formulation (each entry in the constraint matrix is 0 or 1, and the number of nonzero entries in each column is typically small), such a problem with up to 20,000 columns and 100 rows can often be solved very quickly (e.g., a few CPU seconds). By the time $t_i + \tau$, an optimal solution or at least a reasonably good feasible solution, denoted as S_i, to $[\mathrm{SP}_i]$ will have been obtained. The solution S_i is then implemented over the interval $[t_i + \tau, t_{i+1} + \tau]$.

3.3. Generating Columns Used in Formulation $[\mathrm{SP}_i]$ for $i \ge 2$

To be consistent, we follow the notation defined in the previous sections. The columns used in the formulation $[\mathrm{SP}_i]$ (described in §3.2), i.e., the columns in Γ_j for each vehicle type $j \in Q_i$, are generated dynamically over the time interval $[t_{i-1} + \tau, t_i]$. These columns correspond to trips by the vehicles in V_i covering the orders in N_i. As we mentioned earlier, the corresponding trips of these columns start from the initial locations of the corresponding vehicles at time $t_i + \tau$ and end at the depot. (The formulation $[\mathrm{SP}_i]$ is solved in the time interval $[t_i, t_i + \tau]$, as described in §3.2. No new columns are generated in the time interval $[t_i, t_i + \tau]$.)

By the time $t_{i-1} + \tau$ immediately after the solution S_{i-1} of $[\mathrm{SP}_{i-1}]$ is implemented, the initial location and capacity (or remaining capacity) of each vehicle in V_i are known. Hence, all the information about the available vehicles needed for generating the columns to be used in $[\mathrm{SP}_i]$ is known at the very beginning of the time interval $[t_{i-1} + \tau, t_i]$. Among the orders N_i to be considered in the problem $[\mathrm{SP}_i]$, the orders in $N_{i-1} \backslash C_{i-1}$ have already arrived by time $t_{i-1} + \tau$, and hence they are known throughout the time interval $[t_{i-1} + \tau, t_i]$. However, the orders in O_{i-1} (which are the other part of N_i) arrive over the time interval $[t_{i-1}, t_i]$, and we only know part of them at each time point $t \in [t_{i-1} + \tau, t_i]$. In every iteration of our column generation approach, we consider all the orders in $N_{i-1} \backslash C_{i-1}$ and the orders in O_{i-1} that have arrived by the time this iteration is executed.

In our approach, columns are generated by two means: (i) modifying existing columns by inserting orders into the columns or deleting orders from the columns; (ii) creating new columns to cover new orders (i.e., dispatching new vehicles to cover new orders). These are carried out by alternately solving the LP relaxation of a restricted set-partitioning formulation of an underlying static problem and applying fast local-search-based heuristics. LPs are solved to get dual prices of orders, which are then used to guide the local search in the heuristics. The heuristics are used to generate new columns. As noted in §3.2, we do not try to solve the underlying static problem to optimality. Instead, here we try to solve the LP relaxation of the static problem heuristically by column generation combined with local search heuristics. In the following, we describe the details of our column generation procedure, which consists of five steps (Steps 0 through 4) and repeats Steps 1 through 4 after the initialization Step 0.

Step 0: Generating an Initial Set of Columns. In this step, we try to generate a set of columns based on the existing columns used in the formulation $[SP_{i-1}]$ of the previous static problem and a set of single-order columns. For ease of presentation, we assume that the solution S_{i-1} to the problem $[SP_{i-1}]$ is represented by the set of columns that are adopted and implemented at time $t_{i-1}+\tau$. Let R_{i-1} be the subset of the columns (trips) of S_{i-1} that have not been completed by time $t_i+\tau$. By the notation defined in §3.2, U_{i-1} ($U_{i-1} \subseteq V_i$) is the set of the corresponding vehicles that cover the trips in R_{i-1}.

Step 0.1. Generate a set of columns based on the columns in $[SP_{i-1}]$ as follows. Modify all the columns in the formulation $[SP_{i-1}]$, i.e., the columns in $\Gamma = \cup_{j\in Q_{i-1}}\Gamma_j$, one by one as follows. For each column $u \in \Gamma$, remove from u all the orders that belong to C_{i-1} (i.e., the orders picked up during the time interval $[t_{i-1}+\tau, t_i+\tau]$ under solution S_{i-1}), keep the relative sequence of the remaining orders, reschedule the remaining orders, and reassign a vehicle to cover these orders by the following rules:

(i) If $u \in R_{i-1}$, then the column after the orders from C_{i-1} are removed is a valid column. The same schedule (for the remaining orders) and the same vehicle are used for this column.

(ii) If $u \notin R_{i-1}$ and there are remaining orders after the orders from C_{i-1} are removed, reschedule the remaining orders and assign a new vehicle from the depot to cover these orders such that the trip starts at time $t_i+\tau$.

(iii) If $u \notin R_{i-1}$ and there are no remaining orders after the orders from C_{i-1} are removed, discard this column.

Step 0.2. For each order in $N_{i-1}\backslash C_{i-1}$, create a new single-order column by a new vehicle that starts from the depot at time $t_i+\tau$, covers this order, and then returns to the depot.

Let the set of the columns generated in this step by Π_0. The orders covered in these columns are the ones in $N_{i-1}\backslash C_{i-1}$. After this initialization step, go to Step 1 with the input: $\Pi = \Pi_0$ and $M = N_{i-1}\backslash C_{i-1}$.

Step 1: Setting Up a Set-Partitioning-Type Formulation and Solving Its LP Relaxation. Let Π denote all the columns generated so far, and Π_j the set of columns covered by a vehicle type $j \in Q_i$. Let M denote the set of orders covered in the columns in Π. Define the parameters f_u and δ_{lu}, and binary variables x_u, for $u \in \Pi$ and $l \in M$, in the same way as they are used in the formulation $[SP_i]$. Set up the following set-partitioning-type formulation.

$$[SP] \quad \text{Minimize} \sum_{j\in Q_i}\sum_{u\in\Pi_j} f_u x_u \tag{5}$$

$$\text{s.t.} \sum_{j\in Q_i}\sum_{u\in\Pi_j} \delta_{lu} x_u = 1, \quad \text{for } l \in M \tag{6}$$

$$\sum_{u\in\Pi_j} x_u \le 1, \quad \text{for } j = 1, 2, \ldots, q_i - 1 \tag{7}$$

$$x_u \in \{0, 1\}, \quad \text{for } u \in \Pi. \tag{8}$$

The LP relaxation of this formulation, denoted as [LSP], is exactly the same except that constraint (8) is replaced by "$0 \le x_u \le 1$, for $u \in \Pi$." Solve [LSP] by the LP solver of CPLEX and get the optimal dual variable values denoted as follows: π_l for each order $l \in M$, and σ_j for each vehicle type $j \in \{1, 2, \ldots, q_i - 1\}$. After [LSP] is solved, we try to generate new columns both for the orders in M for which there are already existing columns, and for the orders that newly arrive for which there are no existing columns yet. These are done, respectively, in Steps 2 and 3.

Step 2: Generating New Columns for Old Orders $l \in M$. In this step, we try to generate a prespecified number of new columns by modifying the columns in the optimal basis of [LSP] using a two-phase local search procedure. The reason we utilize the columns in the optimal basis of [LSP] is that each trip in the basis has a zero reduced cost, and if such a trip is modified appropriately, it is likely to get new trips with a negative reduced cost. Once we generate new columns with a negative reduced cost, we add these columns to [LSP], which if resolved can yield a lower objective value.

Given a trip from the optimal basis of [LSP], it is modified in two phases. In the first phase, some orders that are not covered by the trip are selected based on some rule and inserted into the trip. In the second phase, some orders in the trip generated in the first phase are selected based on a similar rule and deleted from the trip. The rule used to select an order for insertion or deletion is a greedy strategy based on the cost for inserting or deleting the order and the dual variable value of the order.

Let $\mathbf{B} = \{s(1), s(2), \ldots, s(B)\}$ denote the set of the B columns in the optimal basis of [LSP]. Let $v(k) \in \{1, 2, \ldots, q_i\}$ and $r(k)$ denote the corresponding vehicle type and remaining capacity of the vehicle covering trip $s(k)$, for $k = 1, \ldots, B$. Note that here the remaining capacity of a vehicle is defined as the initial capacity of the vehicle at time $t_i+\tau$. If the vehicle type is q_i (i.e., a vehicle currently at the depot), then the remaining capacity of the vehicle is 1. A vehicle with some other type may have a remaining capacity less than 1. Let $O(s)$ and $W(s)$ denote the set of the orders and total weight of the orders covered in a column s.

Step 2.0. Let the set of new trips already generated be $\Omega = \varnothing$.

Step 2.1. Pick a trip $s(u) \in \mathbf{B}$. Temporarily relax the capacity of the vehicle that covers trip $s(u)$ to $1.5r(u)$. Let $s = s(u)$.

Step 2.2. (*Phase* 1 : *Order Insertion*).

(i) For each order $l \in M\backslash O(s)$ satisfying $W(s) + w_l \le 1.5r(u)$, do the following (a) and (b). If there is no such l, go to Step 2.3.

(a) Find initial trips by inserting order l at each possible position of s.

(b) Improve each initial trip generated in (a) by performing 2-exchange and Or-exchange as described in Kindervater and Savelsbergh (1997), and find a trip s_l that covers all the orders in $O(s) \cup \{l\}$ with the minimum cost. Let g_l be the cost of trip s_l.

(ii) Select order $p \in M\backslash O(s)$ such that $g_p - \pi_p = \min\{g_l - \pi_l \mid \text{for each } l \in M\backslash O(s) \text{ and } W(s) + w_l \le 1.5r(u)\}$, where π_l is the dual variable value of order l in [LSP]. If $W(s) + w_p \le r(u)$ and the reduced cost of s_p (i.e., $g_p - \pi_p - \sum_{l \in s} \pi_l$) is negative, then let $\Omega = \Omega \cup \{s_p\}$. Let $s = s_p$, and repeat Step 2.2.

Step 2.3. (*Phase* 2 : *Order Deletion*).

(i) For each order $l \in O(s)$, remove it from s. Then improve the remaining trip by performing 2-exchange and Or-exchange, and find a trip s_l covering orders in $O(s)\backslash\{l\}$ with the minimum cost. Let g_l be the cost of trip s_l.

(ii) Select order e such that $g_e + \pi_e = \max\{g_l + \pi_l \mid \text{for each } l \in O(s)\}$. If all the orders in s have been deleted, go to Step 2.4. Otherwise, if $W(s) - w_e \le r(u)$ and the reduced cost of s_e (i.e., $g_e + \pi_e - \sum_{l \in s} \pi_l$) is negative, then let $\Omega = \Omega \cup \{s_e\}$. Let $s = s_e$, and repeat Step 2.3 until all the orders in s have been deleted.

Step 2.4. If the number of trips in Ω has already reached a prespecified number (e.g., 1,000), stop. Otherwise, go to Step 2.1.

Step 3: Generating New Columns for New Orders. Let η_1 denote the time at which Step 3 was executed in the last iteration. Let $\eta_1 = t_{i-1}$ if the current iteration is the first iteration, and hence Step 3 has not been executed before. Let η_2 denote the current time, i.e., the time at which Step 3 is executed in the current iteration. If no new orders have arrived during the time period $[\eta_1, \eta_2]$, go to Step 4. Otherwise, let $O[\eta_1, \eta_2]$ denote the set of orders that have arrived over $[\eta_1, \eta_2]$, and execute Steps 3.1 and 3.2. Step 3.1 generates a single-order trip for each new order, and Step 3.2 uses a heuristic rule to generate new columns by inserting each new order into a column in the optimal basis of [LSP].

Step 3.0. Let the set of new trips already generated be $\Psi = \varnothing$.

Step 3.1. For each order $l \in O[\eta_1, \eta_2]$, create a new single-order trip by a new vehicle that starts from the depot at time $t_i + \tau$, covers this order, and returns to the depot. Add these columns to Ψ.

Step 3.2. As in Step 2, here we use the columns in the optimal basis of [LSP] to generate new columns. Following the notation defined in Step 2, let $\mathbf{B} = \{s(1), s(2), \ldots, s(B)\}$ denote the set of the B columns in the optimal basis of [LSP]. Let $v(k) \in \{1, 2, \ldots, q_i\}$ and $r(k)$ denote the corresponding vehicle type and remaining capacity of the vehicle covering trip $s(k)$, for $k = 1, \ldots, B$. For each order $l \in O[\eta_1, \eta_2]$, do the following (i) and (ii).

(i) For each column $s(k) \in \mathbf{B}$ with $r(k) \ge w_l$, generate initial trips by inserting order l at each possible position of $s(k)$.

(ii) Improve each initial trip generated in (i) by performing 2-exchange and Or-exchange and find a trip s_l that results in the minimum increase in cost due to the insertion of order l. Add trip s_l to Ψ.

Step 4: Updating the Formulation [SP]**.** In this step, we expand the formulation [SP] by adding the new columns generated in Steps 2 and 3, i.e., the columns in $\Omega \cup \Psi$, and new rows representing new orders in $O[\eta_1, \eta_2]$. Then go to Step 1 to solve the LP relaxation of the updated formulation [SP]. Steps 2 through 4 are repeated as well.

3.4. Generating Columns Used in Formulation [SP_1]

The columns used in the formulation [SP_1] of the very first static problem at time $t_1 = 0$ are generated before time 0. Because the initial set of orders N_0 are left over from the previous planning horizon, we assume that there is a sufficient amount of time to generate as many columns as necessary for these orders before time 0. (This can be done in early morning before the new planning horizon begins.) Therefore, we do not set a time limit for generating columns for these initial orders. The following steps briefly describe how columns are generated for these initial orders before time 0.

Step 1. For each order in N_0, create a single-order trip by a vehicle that starts from the depot at time t_1, picks up this order, and then returns to the depot. These columns serve as initial columns in the set-partitioning-type formulation of the problem, which is the same as [SP] described in §3.3 except that constraint (7) is removed (because all the vehicles are at the depot at time 0) and M is replaced by N_0. This formulation is solved in Step 2.

Step 2. Apply the column generation approach described in §3.3. To generate new columns, apply the heuristic described in Step 2 of §3.3 with $M = N_0$, and Π being the set of the columns generated in Step 1. Steps 1, 2, and 4 of the approach described in §3.3 are executed until no new columns can be generated.

4. Two Approaches Used for Comparison

In this section, we propose two other algorithms for our problem that will be used to evaluate the performance of the dynamic column generation approach

DYCOL proposed in §3. As we will soon see, these algorithms are allowed to take an unlimited amount of computational time for solving each static problem involved, and hence may not be implementable at all.

These two algorithms, called INS and COL for ease of presentation, follow the same rolling horizon framework as DYCOL, that is, in these algorithms, the planning horizon $[0, H]$ is divided by K evenly spaced decision epochs $t_1, t_2, \ldots, t_K$ with $t_1 = 0$, and $t_1 < t_2 < \cdots < t_K$, and a static problem at each decision epoch t_i is solved. There are two major differences between DYCOL and these two algorithms, as follows:

(i) Computational times allowed to solve static problems are different. In DYCOL, the static problem at each decision epoch t_i is solved over time interval $[t_i, t_i + \tau]$, where τ is the allowed computational time, and the solution is implemented over time interval $[t_i + \tau, t_{i+1} + \tau]$. In INS and COL, at each decision epoch t_i, we assume that the time is frozen and allow an unlimited amount of time to solve the associated static problem. Because we freeze time at each decision epoch t_i, the solution to this static problem is implemented over time interval $[t_i, t_{i+1}]$. The dynamism of these two algorithms is depicted in Figure 2(a). We note that because of the difference in the computational time allowed in each decision epoch, it is not possible to have both the same decision epochs and the same implementation epochs in INS and COL as in DYCOL. However, given that in our computational test, τ (5 or 10 seconds) is a relative small amount of time compared to the time between two consecutive decision epochs (60 seconds), the timeline difference between INS/COL and DYCOL is fairly small.

(ii) Different solution techniques are used to solve static problems. In DYCOL, to solve the static problem at t_i, a column generation approach is used (as described in §3), where columns are generated dynamically over the time interval $[t_{i-1} + \tau, t_i]$, and a set-partitioning-type problem containing all the columns generated by time t_i is solved by the MIP solver of CPLEX. Note that because the columns are generated over time, some columns are generated without knowing all the information up to time t_i. Also, because there is a computational time limit τ, only a limited number of columns can be generated. We use a similar column generation approach for static problems in COL, but columns are generated at time t_i with all the information up to that time point. As we do not set a limit on computational time, in theory, as many columns as necessary can be generated, and hence the static problem at each decision epoch could be solved to optimality. In INS, we use an insertion-based local search heuristic for static problems, and the heuristic is allowed to take an unlimited computational time.

The details of INS and COL are described in the following subsections. We use the same notation defined in §3 with the added assumption that $\tau = 0$. Suppose at t_{i-1} the static problem is solved with the solution S_{i-1}, which is implemented over the interval

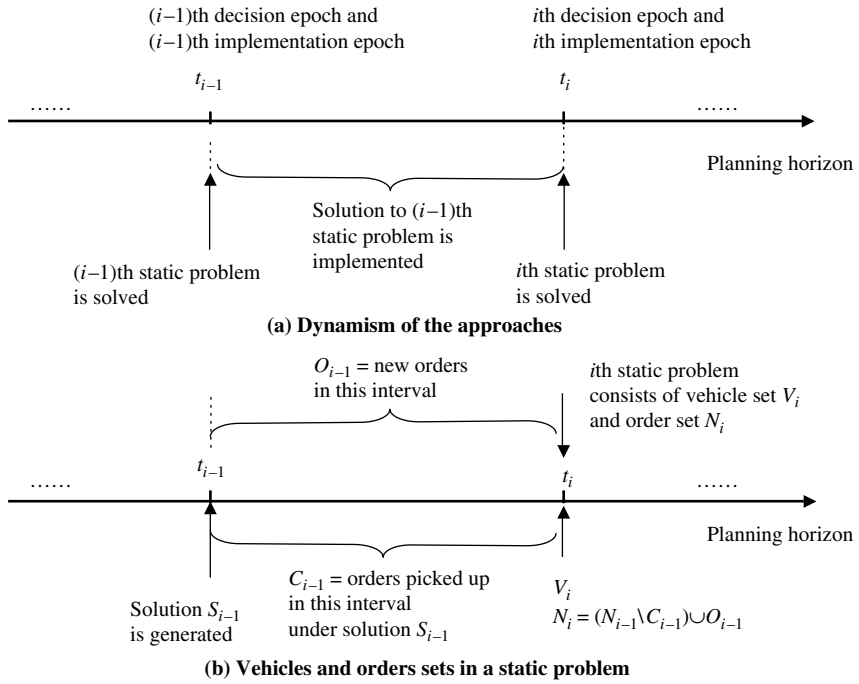

Figure 2 An Illustration of the Structure of the Solution Approaches COL and INS

$[t_{i-1}, t_i]$. During the time interval $[t_{i-1}, t_i]$, a set of new orders O_{i-1} arrives and another set of orders C_{i-1} are picked up under solution S_{i-1}. Then, at t_i, the static problem consists of the order set $N_i = (N_{i-1} \setminus C_{i-1}) \cup O_{i-1}$ and the vehicle set V_i defined as in §3, but with $\tau = 0$. See Figure 2(b) for an illustration of the vehicle and order sets involved in the static problem at t_i in algorithms INS and COL.

4.1. Algorithm INS

This approach solves the static problem at the first decision epoch t_1 by a column generation approach and the static problem at every other decision epoch t_i, for $i = 2, \ldots, K$, by an insertion-based heuristic generalized from the insertion heuristic proposed by Solomon (1987) for the static VRPTW problem. As noted in §1, similar insertion-based heuristics have been commonly used to solve various dynamic routing problems in the literature. Our insertion heuristic tries to insert each new order in O_{i-1} (which has arrived during period $[t_{i-1}, t_i]$) into an en route trip, or to dispatch a new vehicle from the depot to cover it. If there is no new order (i.e., O_{i-1} is empty), it uses local search to improve the current trips. The steps of the algorithm INS are summarized as follows.

Step 0: Generating a Solution to the Static Problem at $t_1 = 0$. At time $t_1 = 0$, we use the column-generation-based approach described in §3.4 to generate as many columns as necessary for the initial orders in N_0. Get a solution by solving the set-partitioning-type formulation $[SP_1]$ formed by these columns by the MIP solver of CPLEX.

Step 1: Inserting New Orders. To construct a solution to the static problem at t_i, we use the trips in the solution S_{i-1} of the previous decision epoch as a starting point. We first remove the orders in C_{i-1} (which have been implemented) from the trips in S_{i-1} and keep their original schedule from time t_i onward. The resulting trips form a valid solution for the orders in $N_{i-1} \setminus C_{i-1}$, which are a subset of the orders involved in the static problem at t_i. Next we try to insert into these trips the other orders involved in the problem, i.e., the orders in O_{i-1} (which are the new orders that have arrived during period $[t_{i-1}, t_i]$). Let S_i denote the set of the trips already generated by modifying the trips in S_{i-1} as described earlier. The following steps describe how each order $l \in O_{i-1}$ is either inserted into the trips in S_i or covered by a new vehicle.

Step 1.1. For each trip $s(k) \in S_i$ with $r(k) \geq w_l$, generate initial trips by inserting order l at each possible position of $s(k)$. Recall that $r(k)$ is the remaining capacity of vehicle $v(k)$.

Step 1.2. Improve each initial trip generated in Step 1.1 by performing 2-exchange and Or-exchange. Find a trip s_l that results in the minimum increase in cost due to the insertion of order l. Let $s(k_l) \in S_i$ be the trip based on which s_l is built. Replace $s(k_l)$ with s_l in S_i, i.e., let $S_i = (S_i \setminus \{s(k_l)\}) \cup \{s_l\}$. Go to Step 1.1 for next order in O_{i-1}.

Step 1.3. If no such feasible s_l exists, then generate a new single-order trip starting from the depot at t_i, covering the order, and then returning to the depot. Add this trip to S_i.

Step 2: Improving S_i by Local Search. If every order $l \in O_{i-1}$ is covered after Step 1, then we have a feasible solution S_i to the static problem at t_i. To improve this solution, we apply local search techniques such as 2-exchange and Or-exchange (e.g., relocating an order from one trip to another trip, exchanging orders in the same trip or between different trips). This is executed repeatedly until no improvement is possible.

4.2. Algorithm COL

Like DYCOL, Algorithm COL also uses a column-generation-based approach for handling the static problem at each decision epoch. However, in COL, all the columns used in the static problem at time t_i are generated at time t_i, and it is allowed to take an unlimited amount of computational time to generate columns and solve the static problem. Thus, the solution obtained by this approach for the static problem at each decision epoch should be very close to or could be an optimal solution of the problem. The procedure for generating columns at each epoch t_i in COL is the same as the algorithm described in §3.3 except that all of the steps involved are executed at time t_i with all the information known by that time, and there is no computational time limit. Thus there is no limit on the number of columns that can be generated in Step 2, and all the new orders in O_{i-1} are considered in Step 3.

5. Computational Results

In this section, we report the computational results on the performance of DYCOL compared to INS and COL. In §5.1, we describe how test problems are generated, and in §5.2, we summarize various testing results.

5.1. Testing Problems

Our testing problems are generated based on Solomon's 100-customer benchmark problems (Solomon 1987, and website: http://web.cba.neu.edu/~msolomon/problems.htm) for the static vehicle routing problem with time windows (VRPTW). In these benchmark problems, 100 customer locations are distributed in a 100×100 square in a Euclidean plane, and the travel times between locations are equal to the corresponding Euclidean distances.

There are six types of problems, named R1, R2, C1, C2, RC1, and RC2, each with 8 to 12 problems. Different types of problems differ in the distribution of customer locations, service time at each customer, and width of time windows as follows:

(i) The customer locations are uniformly distributed in the problems of Types R1 and R2, clustered in the problem of Types C1 and C2, and mixed in the problems of Types RC1 and RC2.

(ii) The service time at each customer is 10 time units in the problems of Types R1, R2, RC1, and RC2; and 90 time units in the problems of Types C1 and C2.

(iii) In each problem, there is a time window $[0, l_0]$ associated with the depot within which a vehicle must return to the depot after covering some customers. The problems of Types R1, C1, and RC1 have a narrow time window at the depot, such that only a few customers can be covered in each trip, while problems of Types R2, C2, and RC2 have a wider time window at the depot.

We generate testing problems by adapting all six types of Solomon's problems. Given a Solomon's problem, we generate 20 different test problems randomly based on the following specifications, which guarantee that every order in a test problem will be covered within its time window in a solution generated by DYCOL, INS, and COL.

(a) The length of the planning horizon H is set to be 1,200 seconds for problems of Types R1, C1, and RC1; and 2,400 seconds for those of Types R2, C2, and RC2.

(b) Each customer in a Solomon's problem corresponds to an order in our test problems with the same location and same weight. Hence, there are exactly 100 orders in each of our test problems, and the distance matrix between orders in our problems is exactly the same as that in the corresponding Solomon's problems. Also, the vehicles in our test problems have the same capacity as those in the Solomon's problems.

(c) Because the time window of the depot $[0, l_0]$ in the Solomon's problem corresponds to the planning horizon $[0, H]$ in our problems, all the time-related data in the Solomon's problems have to be scaled by multiplying by H/l_0. That is, given the time window $[e_i, l_i]$ of customer i, service time s_i at customer i, and travel time t_{ij} from customer i to j in a Solomon's problem, in our problem, the corresponding time window of order i is $[a_i, b_i] = [e_i H/l_0, l_i H/l_0]$, service time of order i is $st_i = s_i H/l_0$, and travel time from order i to j is $t_{ij} H/l_0$. This guarantees that any feasible solution to the Solomon's problem is also feasible to the corresponding dynamic problem. Therefore, the optimal objective value of a Solomon's problem is a lower bound of the optimal objective value of a corresponding dynamic problem.

(d) The call-in time ct_i for order i is uniformly distributed in the interval

$$\left[\frac{1}{2}\min\left\{\frac{He_i}{l_0}, \frac{Hl_i}{l_0} - d_i - \Delta - \tau\right\}, \min\left\{\frac{He_i}{l_0}, \frac{Hl_i}{l_0} - d_i - \Delta - \tau\right\}\right]$$

(which is the interval $[(1/2)\min\{a_i, b_i - d_i - \Delta - \tau\}, \min\{a_i, b_i - d_i - \Delta - \tau\}]$), where $[e_i, l_i]$ is time window of customer i and d_i is the travel time (equal to travel distance) from the depot to customer i in the Solomon's problem, and Δ is the time between two consecutive decision epochs and τ is the computational time assigned to solve $[\mathrm{SP}_i]$ in our approach DYCOL. All the orders with a nonpositive call-in time are put in the initial order set N_0 in our problem. The number of such orders varies from 25 to 75 in a test problem. Therefore, by Larsen, Madsen, and Solomon (2002), all our test problems are *partially* dynamic in the sense that not every order arrives randomly after the beginning of the planning horizon. For any order i with a positive call-in time, because the call-in time is no later than $b_i - d_i - \Delta - \tau$, it is feasible for the dispatcher to send a new vehicle from the depot at the next implementation epoch (which is at most $\Delta + \tau$ time units apart from this call-in time) to cover this order before or at its right time window b_i. This guarantees that there is always a feasible solution to the dynamic problem.

The parameters involved in the solution approaches are generated as follows:

(1) The time between two consecutive decision epochs, Δ, is set to be 60 seconds for problem Types R1, C1, and RC1; and 120 for problem Types R2, C2, and RC2. Because the length of the planning horizon is 1,200 seconds for problem Types R1, C1, and RC1; and 2,400 seconds for problem Types R2, C2, and RC2; there are 20 decision epochs involved in all the three approaches DYCOL, INS, and COL for each test problem.

(2) For DYCOL, the computational time τ dedicated to solving the static problem at each decision epoch is five seconds for problem Types R1, C1, and RC1; and 10 seconds for problem Types R2, C2, and RC2.

5.2. Computational Results

All the algorithms in this paper, DYCOL, COL, INS, are implemented in C++ and run in Windows XP on a PC with one 1.6 GHz CPU. All linear programming problems and set-partitioning-type integer programming problems involved in column generation procedures are solved by LP and MIP solvers of CPLEX, Version 7.0, respectively.

Tables 1 and 2 show the computational results for all six types of problems described in §5.1. Column (1)

Table 1 Testing Results for R1, C1, RC1 Problems

	STATIC				COL	INS		DYCOL		
(1)	Solomon's obj. (2)	Our obj. (3)	Gap (%) (4)	CPU (5)	Obj. (6)	Obj. (7)	Gap^{COL} (%) (8)	Obj. (9)	Gap^{COL} (%) (10)	Gap^{INS} (%) (11)
R101	1,637.7	1,672.64	2.13	17.91	1,688.51	1,702.34	0.82	1,674.54	−0.83	−1.63
R102	1,466.6	1,488.17	1.47	48.97	1,528.97	1,578.98	3.27	1,524.28	−0.31	−3.46
R103	1,208.7	1,231.28	1.87	101.92	1,319.87	1,443.74	9.39	1,341.29	1.62	−7.10
R104	971.5	1,059.41	5.18	1,921.87	1,169.33	1,178.02	0.74	1,162.50	−0.58	−1.32
R105	1,355.3	1,391.57	2.68	29.64	1,471.44	1,507.08	2.42	1,472.64	0.08	−2.29
R106	1,234.6	1,263.42	2.33	82.47	1,355.54	1,436.55	5.98	1,371.78	1.20	−4.51
R107	1,064.6	1,107.76	4.05	191.80	1,230.90	1,326.64	7.78	1,245.17	1.16	−6.14
R108	960.88	1,000.73	4.15	2,575.72	1,089.43	1,149.17	5.48	1,104.96	1.43	−3.85
R109	1,146.9	1,235.52	7.73	1,371.20	1,385.72	1,434.16	3.50	1,325.89	−4.32	−7.55
R110	1,068.0	1,149.82	7.66	760.59	1,252.54	1,373.10	9.63	1,302.55	3.99	−5.14
R111	1,048.7	1,103.16	5.19	95.72	1,239.70	1,313.26	5.93	1,224.65	−1.21	−6.75
R112	982.14	1,044.58	6.36	2,078.27	1,174.77	1,225.01	4.28	1,189.67	1.27	−2.88
C101	827.3	828.94	0.20	32.24	828.94	854.31	3.06	846.72	2.15	−0.89
C102	827.3	829.70	0.29	59.00	931.98	1,051.84	12.86	989.84	6.21	−5.89
C103	826.3	828.07	0.21	72.64	966.07	1,178.88	22.03	1,048.54	8.54	−11.06
C104	822.9	826.67	0.46	163.36	977.04	1,167.56	19.50	1,044.17	6.87	−10.57
C105	827.3	828.94	0.20	47.05	834.79	860.17	3.04	834.79	0.00	−2.95
C106	827.3	829.70	0.29	38.14	979.76	1,113.61	13.66	997.68	1.83	−10.41
C107	827.3	829.70	0.29	45.77	832.21	946.29	13.71	843.49	1.36	−10.86
C108	827.3	828.94	0.20	130.63	1,075.79	1,261.35	17.25	1,134.10	5.42	−10.09
C109	827.3	828.94	0.20	72.61	977.94	1,317.40	34.71	987.85	1.01	−25.02
RC101	1,619.8	1,720.74	6.23	587.03	1,719.52	1,777.24	3.36	1,732.12	0.73	−2.54
RC102	1,457.4	1,544.52	5.98	166.36	1,551.08	1,635.37	5.43	1,586.97	2.31	−2.96
RC103	1,258.0	1,387.35	10.28	3,600.64	1,450.01	1,513.58	4.38	1,471.98	1.52	−2.75
RC104	1,135.48	1,226.77	8.04	1,498.61	1,332.37	1,436.33	7.80	1,320.03	−0.93	−8.10
RC105	1,513.7	1,629.05	7.62	3,602.88	1,654.20	1,848.97	11.77	1,722.70	4.14	−6.83
RC106	1,424.73	1,589.95	11.60	3,600.66	1,563.99	1,619.29	3.54	1,583.92	1.27	−2.18
RC107	1,207.8	1,304.88	6.04	1,057.41	1,472.50	1,516.09	2.96	1,511.53	2.65	−0.30
RC108	1,114.2	1,209.66	6.13	403.95	1,357.55	1,402.66	3.32	1,414.74	4.21	0.86
Average			3.97				8.33		1.82	−5.69

of these tables lists all the Solomon's 100-customer problems: 12 R1, 9 C1, 8 RC1, 11 R2, 8 C2, and 8 RC2 problems. Column (2) is the lowest known objective value of the Solomon's static VRPTW problems reported in the literature (which may or may not be optimal). (See Solomon's website: http://web.cba.neu.edu/~msolomon/problems.htm, viewed on June 10, 2005, shortly before the submission of the first revision of this paper.) In our computational experiment, we first evaluated the performance of the static version of the column generation approach proposed in §3 for the static VRPTW problem by applying it to each of the Solomon's static 100-customer problems. The results are reported in Columns (3), (4), and (5) in these tables. Column (3) is the objective value obtained by the static version of our column generation approach. Column (4) is the relative gap between the objective value obtained by us given in Column (3) (denoted as $Z(3)$) and the lowest known objective value given in Column (2) (denoted as $Z(2)$), i.e., $((Z(3)-Z(2))/Z(2))\times 100\%$. This relative gap represents the quality of the solution generated by our approach for the static problems. Column (5) is the CPU time (in seconds) used by the static version of our column generation approach for the static problem. Based on these results, we can make the following observations on the performance of our column generation approach applied to the static VRPTW problems:

(i) For seven of the Solomon's static problems (R202, R204, R205, R207, R210, R211, RC203), our column generation approach generated a better solution than the best solution reported on Solomon's website. The corresponding entries in Columns (2), (3), and (4) of these seven problems (all in Table 2) are shown in bold. See Xu and Chen (2003) for the details of our solutions to these seven problems.

(ii) Overall, the solutions to the Solomon's static problems obtained by our column generation approach are quite close to the best solutions known so far. The average relative gap between the objective value generated by our approach and the best objective values reported on Solomon's website is 3.97% for the problems with a narrow time window, and 0.54%

Table 2 Testing Results for R2, C2, RC2 Problems

	STATIC				COL	INS		DYCOL		
(1)	Solomon's obj. (2)	Our obj. (3)	Gap (%) (4)	CPU (5)	Obj. (6)	Obj. (7)	Gap^{COL} (%) (8)	Obj. (9)	Gap^{COL} (%) (10)	Gap^{INS} (%) (11)
R201	1,143.2	1,220.83	6.79	2,428.63	1,416.01	1,437.46	1.51	1,388.11	−1.97	−3.43
R202	**1,191.7**	**1,072.80**	**−9.98**	2,331.36	1,286.07	1,290.40	0.34	1,317.18	2.42	2.08
R203	939.54	1,048.00	11.54	2,406.88	1,118.17	1,172.02	4.82	1,256.65	12.38	7.22
R204	**825.52**	**761.54**	**−7.75**	1,239.42	877.92	1,006.96	14.70	959.09	9.25	−4.75
R205	**994.42**	**964.35**	**−3.02**	1,219.80	1,220.34	1,169.02	−4.21	1,218.51	−0.15	4.23
R206	906.14	1,013.22	11.82	2,459.83	1,087.26	1,149.31	5.71	1,062.66	−2.26	−7.54
R207	**893.33**	**843.12**	**−5.62**	1,829.22	1,058.01	1,137.78	7.54	1,026.35	−2.99	−9.79
R208	726.75	729.52	0.38	1,299.63	828.44	952.13	14.93	924.34	11.58	−2.92
R209	909.16	918.87	1.07	2,406.82	1,065.22	1,153.87	8.32	1,153.50	8.29	−0.03
R210	**939.34**	**938.58**	**−0.08**	1,377.49	1,117.57	1,189.26	6.41	1,160.58	3.85	−2.41
R211	**892.71**	**849.03**	**−4.89**	2,431.63	993.78	976.32	−1.76	986.92	−0.69	1.09
C201	589.1	591.56	0.42	317.44	591.56	591.56	0.00	591.56	0.00	0.00
C202	589.1	591.56	0.42	254.67	657.88	834.49	26.85	680.13	3.38	−18.50
C203	588.7	591.17	0.42	635.11	781.14	874.31	11.93	738.26	−5.49	−15.56
C204	588.1	590.60	0.00	795.63	1,016.69	1,043.80	2.67	1,079.88	6.22	3.46
C205	586.4	588.88	0.42	318.58	588.88	588.88	0.00	588.88	0.00	0.00
C206	586.0	588.88	0.49	537.11	588.49	588.49	0.00	588.49	0.00	0.00
C207	585.8	588.29	0.43	651.14	588.29	588.29	0.00	588.29	0.00	0.00
C208	585.8	588.32	0.43	790.97	621.55	623.26	0.28	616.41	−0.83	−1.10
RC201	1,261.8	1,295.82	2.70	2,322.04	1,557.92	1,798.51	15.44	1,543.31	−0.94	−14.19
RC202	1,092.3	1,099.82	0.69	2,278.10	1,431.00	1,594.26	11.41	1,358.05	−5.10	−14.82
RC203	**1,049.62**	**981.24**	**−6.51**	3,662.26	1,257.21	1,371.33	9.08	1,383.49	10.04	0.89
RC204	798.41	806.27	0.98	5,422.05	981.90	1,176.38	19.81	1,050.42	6.98	−10.71
RC205	1,154.0	1,180.66	2.31	3,837.07	1,582.07	1,640.37	3.69	1,460.09	−7.71	−10.99
RC206	1,146.32	1,206.39	5.24	1,233.75	1,328.65	1,427.16	7.41	1,348.57	1.50	−5.51
RC207	1,061.14	1,070.88	0.92	2,434.81	1,262.38	1,330.47	5.39	1,229.82	−2.58	−7.56
RC208	828.14	868.68	4.89	2,468.02	948.26	1,163.19	22.67	1,087.60	14.69	−6.50
Average			0.54				7.22		2.22	−4.35

for the problems with a wider time window, respectively. This means that in COL, the static problem at each decision epoch is solved to optimality or near optimality. Thus, COL can be viewed as an approach that generates and implements an optimal or near-optimal solution for the static problem at each decision epoch. As we mentioned in §1, while solving the static problem at each decision epoch to optimality does not guarantee the optimality of the overall solution, the computational results by Yang, Jaillet, and Mahmassani (2004) on a dynamic vehicle assignment problem show that using optimal approaches for the static problems does generate better overall solutions for their problems than using heuristics. As we will report next, we have a similar result here.

Columns (6) through (11) in Tables 1 and 2 report the performance of our dynamic column generation approach DYCOL, when compared to the two benchmark approaches, COL and INS, based on 20 random test problems generated from the Solomon's static problem shown in Column (1) following the specifications described in §5.1. Columns (6), (7), and (9) are the average objective value of the 20 test problems obtained by COL, INS, and DYCOL, respectively. Column (8) is the average relative gap between INS and COL, i.e., $((Z_{INS} - Z_{COL})/Z_{COL}) \times 100\%$. Columns (10) and (11) are the average relative gap between DYCOL and COL, i.e., $((Z_{DYCOL} - Z_{COL})/Z_{COL}) \times 100\%$, and the average relative gap between DYCOL and INS, i.e., $((Z_{DYCOL} - Z_{INS})/Z_{INS}) \times 100\%$, respectively. Here Z_{DYCOL}, Z_{INS}, and Z_{COL} are the objective value obtained by DYCOL, INS, and COL, respectively. Based on these results, we can draw the following conclusions:

(i) On average over all the test problems, the objective value of the solution generated by DYCOL is about 2% more than that obtained by COL. On the other hand, COL does not always perform better than DYCOL, although COL is allowed to take more computational time than DYCOL for the static problem at each decision epoch. In fact, in about 30% of the test problems, DYCOL outperforms COL. This shows that for a dynamic problem, solving the static problem at each decision epoch optimally or nearly optimally does not necessarily guarantee an optimal solution over the entire planning horizon.

(ii) DYCOL performs better than INS on most test problems. The average relative gap between INS and DYCOL is about 5% over all the test problems, and is more than 10% for many problems. Based on this,

it is safe to conclude that, on average, approaches like DYCOL that use optimization-based algorithms to handle the underlying static problems perform better than approaches like INS that use local search heuristics for the static problems. We also note that there are several R2 problems for which INS performs better than DYCOL, and frankly, we do not know why INS performs better than DYCOL over those problems.

6. Conclusions

In this paper, we have studied a dynamic vehicle routing problem with time windows. We have proposed a dynamic column generation approach to solve the problem. The approach is adapted from the classical column generation for static problems. The computational results show that our approach is capable of generating satisfactory solutions in a simulated real-time environment.

Some related topics deserve more research. The decision epochs are fixed and known in advance in our current approach. It will be interesting to see what happens if the decision epochs are determined over time based on the arrival process of orders. Also, one can investigate the possible impact of the way the decision epochs are selected on the solution quality. Another interesting topic for future research is to evaluate the impact of future information on solution quality. In our current approach, we have assumed that there is no future information available, and hence at each decision epoch we only use available information up to that point of time. Now if there is some information available regarding future orders (e.g., probability distribution of call-in times), then the question is how one can incorporate such information into the static problem at each decision epoch to get a better overall solution.

Acknowledgments

The authors would like to thank the referees for their helpful comments and suggestions on an earlier version of the paper. This research was supported in part by the National Science Foundation under Grants CMS-0085658, DMI-0196536, and DMI-0421637.

References

Bard, J. F., G. Kontoravdis, G. Yu. 2002. A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Sci.* **36** 250–269.

Bent, R. W., P. Van Hentenryck. 2004. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Oper. Res.* **52** 977–987.

Bertsimas, D. J., D. Simchi-Levi. 1996. A new generation of vehicle routing research: Robust algorithms, addressing uncertainty. *Oper. Res.* **44** 286–304.

Desrochers, M., J. Desrosiers, M. Solomon. 1992. A new optimization algorithm for the vehicle routing problem with time windows. *Oper. Res.* **40** 342–354.

Fu, L., S. Teply. 1999. On-line and off-line routing and scheduling of dial-a-ride paratransit vehicles. *Comput.-Aided Civil Infrastructure Engrg.* **14** 309–319.

Gendreau, M., J.-Y. Potvin. 1998. Dynamic vehicle routing and dispatching. T. Crainic, G. Laporte, eds. *Fleet Management and Logistics*. Kluwer, Norwell, MA, 115–126.

Gendreau, M., F. Guertin, J. Y. Potvin, É. Taillard. 1999. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Sci.* **33** 381–390.

Godfrey, G., W. B. Powell. 2002a. An adaptive dynamic programming algorithm for dynamic fleet management, I: Single period travel times. *Transportation Sci.* **36** 21–39.

Godfrey, G., W. B. Powell. 2002b. An adaptive dynamic programming algorithm for dynamic fleet management, II: Multiperiod travel times. *Transportation Sci.* **36** 40–54.

Ichoua, S., M. Gendreau, J. Y. Potvin. 2000. Diversion issues in real-time vehicle dispatching. *Transportation Sci.* **34** 426–438.

Kindervater, G. A. P., M. W. P. Savelsbergh. 1997. Vehicle routing: Handling edge exchanges. E. H. L. Aarts, J. K. Lenstra, eds. *Local Search in Combinatorial Optimization*. Wiley, Chichester, UK, 337–360.

Kohl, N., J. Desrosiers, O. B. G. Madsen, M. M. Solomon, F. Soumis. 1999. 2-path cuts for the vehicle routing problem with time windows. *Transportation Sci.* **33** 101–116.

Larsen, A., O. B. G. Madsen, M. M. Solomon. 2002. Partially dynamic vehicle routing—Models and algorithms. *J. Oper. Res. Soc.* **53** 637–646.

Larsen, A., O. B. G. Madsen, M. M. Solomon. 2004. The a-priori dynamic traveling salesman problem with time windows. *Transportation Sci.* **38** 459–572.

Mahmassani, H. S., Y. Kim, P. Jaillet. 2000. Local optimization approaches to solve dynamic commercial fleet management problems. *Transportation Res. Record* **1733** 71–79.

Powell, W. B. 1988. A comparative review of alternative algorithms for the dynamic vehicle allocation problem. B. L. Golden, A. A. Assad, eds. *Vehicle Routing: Methods and Studies*. Elsevier Science Publishers, Amsterdam, The Netherlands, 249–291.

Powell, W. B. 1996. A stochastic formulation of the dynamic assignment problem, with an application to truckload motor carriers. *Transportation Sci.* **30** 195–219.

Powell, W. B., T. Carvalho. 1998. Dynamic control of logistics queueing networks for large scale fleet management. *Transportation Sci.* **32** 90–109.

Powell, W. B., P. Jaillet, A. Odoni. 1995. Stochastic and dynamic networks and routing. M. O. Ball, T. L. Magnanti, C. L. Monma, G. L. Nemhauser, eds. *Network Routing*, Vol. 8. *Handbooks in Operations Research and Management Science*. North-Holland, Amsterdam, The Netherlands, 141–295.

Psaraftis, H. N. 1995. Dynamic vehicle routing: Status and prospects. *Ann. Oper. Res.* **61** 143–164.

Regan, A. C., H. S. Mahmassani, P. Jaillet. 1996. Dynamic decision making for commercial fleet operations using real-time information. *Transportation Res. Record* **1537** 91–97.

Regan, A. C., H. S. Mahmassani, P. Jaillet. 1998. Evaluation of dynamic fleet management systems simulation framework. *Transportation Res. Record* **1645** 176–184.

Shen, Y., J.-Y. Potvin, J.-M. Rousseau, S. Roy. 1995. A computer assistant for vehicle dispatching with learning capabilities. *Ann. Oper. Res.* **61** 189–211.

Shieh, H. M., M. D. May. 1998. On-line vehicle routing with time windows, optimization-based heuristics approach for freight demands requested in real-time. *Transportation Res. Record* **1617** 171–178.

Solomon, M. M. 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.* **35** 254–265.

Xu, H., Z.-L. Chen. 2003. A note on Solomon's benchmark VRPTW problems. Working paper, R. H. Smith School of Business, University of Maryland, College Park, MD.

Yang, J., P. Jaillet, H. Mahmassani. 1999. On-line algorithms for truck fleet assignment and scheduling under real-time information. *Transportation Res. Record* **1667** 107–113.

Yang, J., P. Jaillet, H. Mahmassani. 2004. Real-time multi-vehicle truckload pick-up and delivery problems. *Transportation Sci.* **38** 135–148.