# REAL-TIME SCHEDULING APPROACHES FOR VEHICLE-BASED INTERNAL TRANSPORT SYSTEMS

Tuan Le-Anh and M.B.M. De Koster



| ERIM REPORT SERIES *RESEARCH IN MANAGEMENT* | |
|---|---|
| ERIM Report Series reference number | ERS-2004-056-LIS |
| Publication | July 2004 |
| Number of pages | 27 |
| Email address corresponding author | ltuan@fbk.eur.nl |
| Address | Erasmus Research Institute of Management (ERIM)<br>Rotterdam School of Management / Rotterdam School of Economics<br>Erasmus Universiteit Rotterdam<br>P.O. Box 1738<br>3000 DR Rotterdam, The Netherlands<br>Phone: +31 10 408 1182<br>Fax: +31 10 408 9640<br>Email: info@erim.eur.nl<br>Internet: www.erim.eur.nl |

Bibliographic data and classifications of all the ERIM reports are also available on the ERIM website: www.erim.eur.nl

ERASMUS RESEARCH INSTITUTE OF MANAGEMENT

REPORT SERIES

*RESEARCH IN MANAGEMENT*

| BIBLIOGRAPHIC DATA AND CLASSIFICATIONS | | |
|---|---|---|
| Abstract | In this paper, we study the problem of scheduling and dispatching vehicles in vehicle-based internal transport systems within warehouses and production facilities. We develop and use two rolling horizon policies to solve real-time vehicle scheduling problems. To solve static instances of scheduling problems, we propose two new heuristics: combined and column-generation heuristics. We solve a real-time scheduling problem by applying a heuristic to dynamically solve a series of static instances under a rolling horizon policy. A rolling horizon can be seen either as a fixed-time interval in which advance information about loads' arrivals is available, or as a fixed number of loads which are known to become available in the near future. We also propose a new look-ahead dynamic assignment algorithm, a different dynamic vehicle-scheduling approach. We evaluate these dynamic scheduling strategies by comparing their performance with that of two of the best online vehicle dispatching rules mentioned in the literature. Experimental results show that the new look-ahead dynamic assignment algorithm and dynamic scheduling approaches consistently outperform vehicle dispatching rules. | |
| Library of Congress Classification (LCC) | 5001-6182 | Business |
| | 5201-5982 | Business Science |
| | HB 143.7 | Optimization techniques |
| Journal of Economic Literature (JEL) | M | Business Administration and Business Economics |
| | M 11<br>R 4 | Production Management<br>Transportation Systems |
| | C 61 | Optimization techniques, Programming Models |
| European Business Schools Library Group (EBSLG) | 85 A | Business General |
| | 260 K<br>240 B | Logistics<br>Information Systems Management |
| | 255 G | Operations research |
| Gemeenschappelijke Onderwerpsontsluiting (GOO) | | |
| Classification GOO | 85.00 | Bedrijfskunde, Organisatiekunde: algemeen |
| | 85.34<br>85.20 | Logistiek management<br>Bestuurlijke informatie, informatieverzorging |
| | 85.03 | Methoden en technieken, operations research |
| Keywords GOO | Bedrijfskunde / Bedrijfseconomie | |
| | Bedrijfsprocessen, logistiek, management informatiesystemen | |
| | Intern vervoer, scheduling, modellen | |
| Free keywords | Vehicle dispatching and scheduling, vehicle-based internal transport system, real-time scheduling | |

# REAL-TIME SCHEDULING APPROACHES FOR VEHICLE-BASED INTERNAL TRANSPORT SYSTEMS

Tuan Le-Anh*, MBM De Koster

*Rotterdam School of Management, Erasmus University Rotterdam, P.O. Box 1738, 3000 DR Rotterdam, The Netherlands*

**Corresponding author. Tel.: +31 10 4082027; fax: + 31 10 408 9014; e-mail LTuan@fbk.eur.nl*

**Abstract**

In this paper, we study the problem of scheduling and dispatching vehicles in vehicle-based internal transport systems within warehouses and production facilities. We develop and use two rolling horizon policies to solve real-time vehicle scheduling problems. To solve static instances of scheduling problems, we propose two new heuristics: combined and column-generation heuristics. We solve a real-time scheduling problem by applying a heuristic to dynamically solve a series of static instances under a rolling horizon policy. A rolling horizon can be seen either as a fixed-time interval in which advance information about loads' arrivals is available, or as a fixed number of loads which are known to become available in the near future. We also propose a new look-ahead dynamic assignment algorithm, a different dynamic vehicle-scheduling approach. We evaluate these dynamic scheduling strategies by comparing their performance with that of two of the best online vehicle dispatching rules mentioned in the literature. Experimental results show that the new look-ahead dynamic assignment algorithm and dynamic scheduling approaches consistently outperform vehicle dispatching rules.

## 1 Introduction

In many facilities such as modern warehouses, distribution centers, transshipment terminals or manufacturing systems, vehicle-based internal transport (VBIT) systems, particularly automated guided vehicle (AGV) systems, are used to control vehicles. Generally, system controllers dispatch vehicles (or AGVs) using very simple and intuitive online dispatching rules such as nearest-vehicle-first (*NVF*) (Egbelu and Tanchoco, 1984). An important practical reason for selecting simple vehicle dispatching rules is that they are easy to adapt for warehouse management (WMS) or shop-floor control systems (SFC). Also, the dynamic and stochastic environments in which these vehicles have to operate makes a vehicle dispatching approach more obvious than a scheduling approach. Still, a vehicle scheduling approach with a rolling horizon might lead to a better overall system performance than a

dispatching approach. However, this has not been adequately investigated in literature. In this paper we will make such a comparison. The objective of the scheduling problem is minimizing the average load waiting time. Main characteristics of the scheduling problem in real-life VBIT systems are high traffic density, short planning horizon due to stochastic load arrivals and possible vehicle interference problems. These characteristics make offline schedules useless.

Mathematically, the scheduling problem of a VBIT system can be formulated as a pick-up and delivery problem with time windows (PDPTW), in which a vehicle picks up loads at several locations and delivers them to their destinations satisfying certain time-window restrictions. Since in many VBIT systems, vehicles are homogenous and have single-load capacity (a vehicle can pick-up only one load at a time), pick-up and delivery points can be represented as one node, which allows us to formulate the VBIT scheduling problem as a multiple traveling salesman problem with time windows (*m*-TSPTW).

The *m*-TSPTW belongs to the class of *NP-Hard* problems, so it is unlikely that we can solve real-life size instances with many vehicles, loads and pick-up and drop-off locations using a commercial optimization package. In this research, we propose three heuristics for solving static (offline) instances of the scheduling problem which are later applied with rolling horizons. The first one is a simple insertion heuristic (Van der Meer, 2000). The second one uses the initial solution created by the insertion heuristic and then applies several improvement algorithms (*Re-insertion*, *Exchange*, *Relocation*) sequentially to improve the solution. The third one bases on the column-generation approach. To apply the third heuristic, we reformulate the scheduling problem as a set-partitioning (-covering) model to select a set of vehicle routes covering all jobs with minimum average job (load) waiting time. In VBIT systems, we prefer using the term load instead of job. Vehicle routes are generated dynamically by solving a shortest-path problem with time-windows (SPPTW). We apply the generalized permanent labeling algorithm (Desrochers and Soumis, 1988) with a slight modification to solve the SPPTW problem. Because of the stochasticity of the VBIT scheduling problem, we should not schedule vehicles too far in advance even in case we have some information about future loads' arrivals. Scheduling based on incomplete (not all loads known, for example), or inaccurate information may lead to frequent rescheduling ("system nervousness") or suboptimal schedules (Van der Meer, 2000). We propose two dynamic rolling horizon approaches to cope with real-time scheduling problems. The first is a traditional time rolling horizon in which vehicles are scheduled for a tentative rolling horizon length ($H$) and only loads arriving within a shorter horizon ($h = aH$, $a < 1$) are considered as

permanent. A new schedule for vehicles is generated every *h* time periods. We also introduce a second rolling horizon approach based on the number of loads (jobs). Under this approach, we make a tentative vehicle schedule with a fixed number of loads (*M*) supposed to arrive in near future. Similar to the rolling by time approach, only the first *m* loads ($m = \lceil a*M \rceil$, $a < 1$) in the tentative schedule are considered as permanent. This policy limits the size of the scheduling problem to a fixed-size that can be solved in reasonable time for real-life applications. In addition to the three rolling horizon heuristics, we propose a look-ahead dynamic assignment algorithm which is simple and fast. Actually, it is a special type of scheduling by time rolling horizon in which offline instances are solved by the assignment algorithm.

We then evaluate the performance of the proposed real-time scheduling approaches and compare their performance with two of the best-performing dispatching rules (*NVF* and *NVF* with a look-ahead period - *NVF_LA*) for our experimental environments. We will show that the real-time dynamic scheduling strategies consistently outperform dispatching rules under various working conditions. The main contributions of our study rely on practical perspectives of solution approaches. We propose a rolling schedule, which can be very useful in practice. We find that a significant improvement is possible only when we have sufficient information about future loads (about three loads or more per vehicle). Our new column-generation heuristic provides a superior performance. The new combined heuristic and the look-ahead dynamic algorithm result in a significant improvement, compared to a simple insertion heuristic, without increasing the running-time.

The next section discusses literature related to vehicle scheduling problems. Section 3 describes the problem formulation and its characteristics. In section 4, we propose solution approaches for static and real-time scheduling problems, and we also provide a performance evaluation for the proposed heuristics. In section 5, we describe experimental environments and parameters and evaluate performance of the proposed dynamic scheduling approaches and two vehicle dispatching rules. Finally in section 6, we draw conclusions and suggest some fruitful future research directions.

## 2 Review of literature

As mentioned before, the scheduling problem for VBIT systems can be seen as a PDPTW problem or in a particular case as an *m*-TSPTW problem. In the literature, the PDPTW, *m*-

TSPTW and vehicle routing problems with time windows (VRPTW) have been studied since long (Desrochers et al., 1988; Savelsbergh and Sol, 1995). Desrochers et al. (1988) provide a review on vehicle routing with time windows including PDPTW and *m*-TSPTW and solutions approaches. Savelsbergh and Sol (1995) focus on PDPTW (referred to as general pickup and delivery problem – GPDP) and their dynamic versions. In their paper, the *m*-TSPTW is referred as the full truckload PDPTW.

As mentioned in Desrochers et al. (1988), there are two main types of optimization algorithms for VRPTW: dynamic programming and branch-and-bound. Both methods are very time consuming and cannot solve practical problems within an acceptable time limit. Dumas et al. (1991) introduce an exact algorithm to solve PDPTW using a column-generation scheme. The sub-problem (or pricing) is a constrained shortest-path problem. Their algorithm can handle multiple depots and different vehicle types. Desaulniers et al. (1998) propose a similar approach to solve multi-depot vehicle scheduling problems with time windows and waiting costs. In order to solve practical-size problems, they also propose a heuristic to speed up the branch-and-bound process. Savelsbergh and Sol (1998) and Xu et al. (2003) propose some adaptation approaches for speeding up the column-generation algorithm. They use several heuristics to generate columns with negative reduced costs and eliminate unattractive columns by sophisticated column management schemes. Besides set-partitioning and column-generation approaches, several other heuristics have been proposed for the VRPTW, such as a saving heuristic (Kindervater and Savelsbergh, 1992; Laporte et al., 2000; Cordeau et al., 2002).

In the literature, studies on the dynamic VRPTW are not as abundant as studies on the static VRPTW. Psaraftis (1988) provides a survey on solution approaches for dynamic vehicle routing problems. Two main approaches include an adaptation of static solution using local operations and an implementation of static algorithms under a rolling horizon. Savelsbergh and Sol (1998) use the rolling horizon approach to solve a dynamic PDPTW. Yang et al. (2004) studied a dynamic truckload PDP. They propose several benchmark local policies that are actually similar to vehicle online dispatching rules in VBIT systems. They also propose two re-optimization policies (MYOPT and OPTUN) to solve the problem dynamically. The MYOPT policy solves a static instance at every step (when information about a new job arrival is received). OPTUN is different from MYOPT by including some opportunity costs which are based on probabilistic knowledge of future requests in the optimization model. The probabilistic knowledge of future requests help to improve the solution quality. They prove that two re-optimization policies outperform local policies. Yang et al. (2004) used CPLEX

to solve the static problem with a 20 seconds time limit. However with such short time limit, CPLEX normally cannot provide a very good (average) result. Fleischmann et al. (2004) use an assignment algorithm to assign jobs to vehicles with the main objectives to minimize the total order delays and vehicle empty travel time. They show that their approach is superior to dispatching rules and some insertion algorithms. In spite of having different objectives, their method is interesting to us, so we adapt their approach for our problems and test its performance.

De Koster et al. (2004) have carried out extensive simulation experiments to compare the performance (average load waiting time and maximum load waiting time) of several good dispatching rules in literature. They show that, for three practical internal transport environments (a warehouse, a production plant and a transshipment terminal), distance-based dispatching rules such as *NVF* outperform other rules. They also show that little information about future loads has a very positive impact to reduce the average load waiting time.

We compare the performance of the rolling horizon scheduling and assignment approach with that of the best vehicle dispatching rules in the study of De Koster et al. (2004).

## 3 Problem description and formulation

For static scheduling of a VBIT system, we define a set of available vehicles ($K$) and a set of jobs ($N$) which need be picked-up within time-windows $[e_p, l_p]$ ($p \in N$) and dropped-off at their delivery locations. The scheduling problem for VBIT systems can be formulated as a PDPTW. However, we reformulate this problem as an $m$-TSPTW by projecting time-windows at delivery locations to the corresponding pick-up locations (assuming a certain transport time) and logically considering a pick-up and a corresponding delivery job as a single job-node. If the time-window at pick-up location is $[e_p, l_p]$, at delivery location is $[e_d, l_d]$, and the travel time between the two locations is $t_{pd}$, the time-window of the job-node will be $[e_n, l_n]$ with $e_n = e_p$, $l_n = \min(l_p, l_d - t_{pd})$. We suppose that the time-window projection for job-nodes is always feasible ($[e_n, l_n] \neq \varnothing$). In many VBIT systems, only one-sided time-windows are present at pick-up locations (load release times, or $r_p$) and no time-windows are present at delivery locations, so $[e_n, l_n]$ is always $\neq \varnothing$. The travel time from job-node $i$ to job-node $j$ ($t_{ij}$) equals the travel time from the origin of job $i$ ($i^+$) to the destination of $i$ ($i^-$) ($t_{i^+i^-}$) plus the travel time from the destination of $i$ to the origin of $j$ ($t_{i^-j^+}$).

The *m*-TSPTW can be seen as a graph $G = (V, A)$, in which $V$ is a set of vertices and $A$ is a set of arcs. $V = \{0\} \cup N \cup \{n+1\}$, where $\{0\}(\{n+1\})$ denotes the depot (end depot) and $N = \{1..n\}$ is the set of (job-)nodes. $A = \{0\} \times N \cup I \cup N \times \{n+1\}$, where $I \subset N \times N$ is the set of arcs connecting job-nodes. $\{0\} \times N$ contains the arcs from the depot to job-nodes and $N \times \{n+1\}$ contains the arcs from job-nodes to end depot (which is the same physical location as the depot in our computations). For each arc $(i,j) \in A$, there is an associated travel time (distance) $t_{ij}$ and for each job-node $i$ there is an associated time-window $[e_i, l_i]$. $K$ is the number of vehicles and $B$ is a big number.

Decision variables are:

- $x_{ij}^k$ ($(i,j) \in A$, $k \in K$) equals 1 if arc $(i,j)$ is covered by vehicle $k$ and 0 otherwise.
- $D_i$ ($i \in N$) indicates the service start time of (job-)node $i$.

In practice, operating areas in VBIT environments such as warehouses are condensed, so the vehicle travel distance criterion becomes less important. In addition in VBIT systems, the number of vehicles is estimated at tactical level (Le-Anh and De Koster, 2004), so the scheduling problem at operational level does not take it into account. These characteristics leave minimizing the average (or total = $\sum_{i \in N} (D_i - e_i)$) load waiting time as the most important objective of the VBIT scheduling problem in practice.

The model formulation becomes then:

*minimize* $\sum_{i \in N} (D_i - e_i)$

*subject to*

$$\sum_{k \in K} \sum_{j \in N} x_{ij}^k = 1 \qquad \forall i \in N \qquad (1)$$

$$\sum_{j \in V} x_{ij}^k - \sum_{j \in V} x_{ji}^k = 0 \quad \forall i \in N, \forall k \in K \quad (2)$$

$$\sum_{j \in N} x_{0j}^k = 1 \qquad \forall k \in K \qquad (3)$$

$$\sum_{i \in N} x_{i,n+1}^k = 1 \qquad \forall k \in K \qquad (4)$$

$$D_i + t_{ij} - D_j \le B\left(1 - x_{ij}^k\right) \qquad \forall i, j \in N, \forall k \in K \quad (5)$$

$$D_0^k + t_{0j} - D_j \le B\left(1 - x_{0j}^k\right) \qquad \forall j \in N, \forall k \in K \quad (6)$$

$$D_i + t_{i,n+1} - D_{n+1}^k \le B\left(1 - x_{i,n+1}^k\right) \quad \forall i \in N, \forall k \in K \quad (7)$$

$$e_i \le D_i \le l_i \qquad \forall i \in V \qquad (8)$$

$$x_{ij}^k \quad \textit{binary} \qquad \forall i, j \in V, \forall k \in K \quad (9)$$

Constraints (1)-(4) form a multi-commodity flow formulation. Constraints (5)-(8) ensure feasibility of the schedule. (9) is the set of binary constraints. This problem (*m*-TSPTW) is a special case of VRPTW, so it also belongs to the class of *NP-Hard* problems (Kindervater and Savelsbergh, 1992) that are difficult to solve. The size of a static instance is about 6

vehicles, 4 loads per vehicle for a typical warehouse, and it can be much bigger, for example, for transshipment terminals (> 25 vehicles) (De Koster et al., 2004).

# 4 Solution approaches

## 4.1 The static scheduling problem

In the previous section, we have formulated the static (or offline) scheduling problem in a VBIT system as an *m*-TSPTW. Generally, we can use general-purpose optimization packages such as CPLEX to solve the *m*-TSPTW. However, such software can solve only small instances of the *m*-TSPTW, which makes them unusable for practical problems. In this section, we describe several heuristics which will be used later to cope with realistic *m*-TSPTW. Some of them have been introduced originally for the TSP and VRP, but they are useful for our research as well. We also propose a new column-generation heuristic, and a new combined heuristic (a combination of existing heuristics). We define the *cost* of a vehicle tour is the average load waiting time of this tour.

### 4.1.1 Insertion heuristic

The insertion heuristic (Van der Meer, 2000; Laporte et al., 2000) is frequently used for real-time dynamic scheduling problems (Psaraftis, 1988). The main advantage of the insertion algorithm is that it is simple and fast. Since, at the decision points in VBIT systems, usually little information about load arrivals is known, rescheduling based on a previous solution does not lead to good solutions. Therefore in our implementation, we rebuild all routes from scratch.

The pseudo code of the insertion algorithm (*Insertion*) is given as follow:

- *Step 0*: Initialize all vehicle routes at the depot node {0}, let the set *S* contain all (job-) nodes arranged in increasing order of the load (job) release times ($S \neq \varnothing$), set all tours' costs to zero.
- *Step 1*: Remove the first node from the set *S* and insert it into a specific tour with least cost, respecting the time-window constraints (5) - (8). By doing this we expand vehicle tours gradually.
- *Step 2*: Repeat step 1 until $S = \varnothing$, compute total cost, stop.

### 4.1.2 Combined heuristic

This heuristic starts with an initial solution created by the insertion heuristic and applies several improvement algorithms sequentially to improve the solution. Three improvement algorithms used in this paper are *Re-insertion*, *Exchange* and *Relocation* (Kindervater and Savelsbergh, 1992; Laporte et al., 2000). We only apply these improvement algorithms and not other more complicated ones, since for the dynamic scheduling approach we do not take too many loads (jobs) into account at once. At each step, we schedule up to about four loads for each vehicle, so other more complicated and time-consuming improvement heuristics such as *k*-opt, with $k \geq 2$, will not be very useful. Among these three algorithms, *Re-insertion* belongs to the class of route improvement heuristics and the two others belong to the class of assignment improvement heuristics. Figure 1 illustrates the three improvement heuristics.

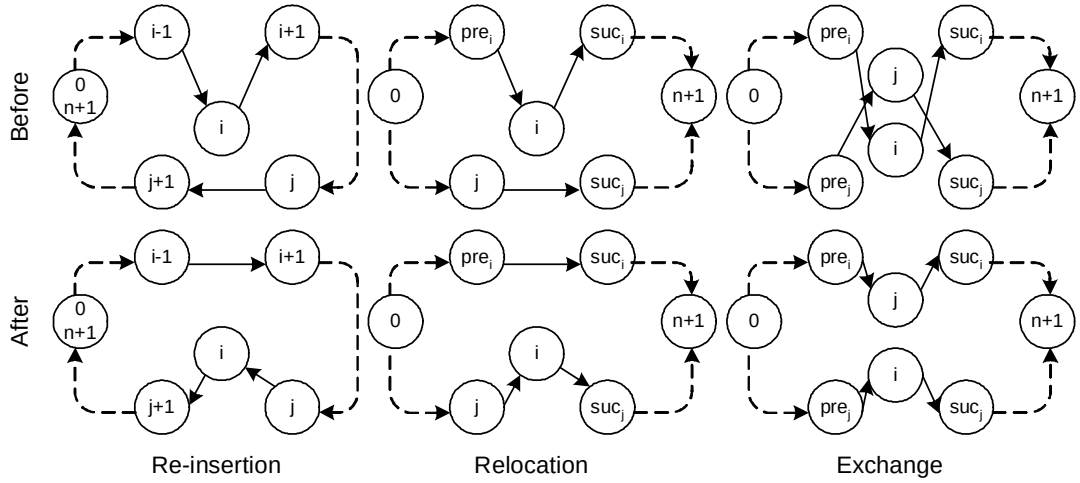



**Figure 1** Improvement heuristic illustrations (Kindervater and Savelsbergh, 1992)

<u>*Re-insertion*</u>: The *Re-insertion* (or forward *Or-exchange*) algorithm works as follow:

- *Step 0*: set *it* (index) to 1 (0 is the depot node).
- *Step 1*: remove the node at position *it* and search for the best insert position from node *it* + 1 to the end of the route.
- *Step 2*: if a cost reduction is found, then insert this node into the best insert position, otherwise increase *it* by 1.
- *Step 3*: if node *it* is the last node in the route, stop. Otherwise go to *Step 1*.

(Node) <u>*Exchange*</u> (between routes):

- *Step 0*: set $it_1$ to 1 (node index for route 1), set *previous total cost* to total cost of route 1 and route 2.
- *Step 1*: find the best exchange position of node $it_1$ and a node in route 2.

- *Step 2*: if a cost reduction is found (*total cost of two new routes* < *previous total cost*), exchange node $it_1$ of route 1 with the best exchange node in route 2.
- *Step 3*: increase $it_1$ by 1, re-compute total route cost, set *previous total cost* to the new total route cost.
- *Step 4*: if all nodes in route 1 have been investigated, stop. Otherwise go to *Step 1*.

(Node) <u>*Relocation*</u> (between routes):

- *Step 0*: set $it_1$ to 1 (node index for route 1), set *previous total cost* to total cost of route 1 and route 2.
- *Step 1*: find the best insert position of node $it_1$ in route 2.
- *Step 2*: if a cost reduction is found (*total cost of two new routes* < *previous total cost*), insert node $it_1$ of route 1 at the best insert position in route 2.
- *Step 3*: increase $it_1$ by 1, re-compute total route cost, set *previous total cost* to the new total route cost.
- *Step 4*: if all nodes in route 1 have been investigated, stop. Otherwise go to *Step 1*.

**General framework for the combined heuristic**

We propose a combined heuristic combining insertion and the improvement algorithms into one combined heuristic. The general-framework for the combined heuristic is given below:

- *Step 0*: create initial (vehicle) routes using the *Insertion* algorithm,
- *Step 1*: applying *Re-insertion* algorithm for initial routes,
- *Step 2*: applying *Exchange* algorithm for every pair of routes of the previous step,
- *Step 3*: applying *Relocation* algorithm for every pair of routes of the previous step,
- *Step 4*: applying *Re-insertion* algorithm again for all routes of the previous step. STOP.

**Complexity of the combined heuristic**

Taking the implementation framework of the combined heuristic into account, we can evaluate the complexity of this heuristic. Obviously, the complexity of the *Insertion* algorithm is $O(n^2)$ ($n$ is the number of loads) (Van der Meer, 2000). Kindervater and Savelsbergh (1992) show that the complexity of the three improvement algorithms are $O(m^2)$ ($m$ is the number of loads per vehicle, $m \leq n$), which is $O(n^2)$ in the worst case. According to the above framework, we apply *Re-insertion* for all routes, so the worst case complexity is $O(kn^2)$ ($k$ is the number of vehicles). Two other improvement algorithms are applied for all pair of routes. The number of route pairs equals $k(k-1)/2$, so the worst case complexity of each assignment improvement algorithm applying for all pairs of routes is $O(k^2n^2)$. Finally,

the overall complexity of the combined algorithm is $O(k^2n^2)$ in the worst case. However, this complexity is rarely the case. Normally, for our scheduling problems, the number of loads served by a vehicle is about the same, so $m \cong n/k$. Hence, the average complexity is $O(k^2m^2) \cong O(n^2)$. Therefore, the complexity of the combined heuristic does not increase much in comparison with the insertion heuristic, and it is fast for practical problems.

### 4.1.3 Column generation heuristic

The column-generation approach has been used by several authors for solving the PDPTW (Dumas et al., 1991; Savelsbergh and Sol, 1998). In this study, we apply this approach to solve the *m*-TSPTW. In order to apply the column generation heuristic we re-formulate the *m*-TSPTW as a set-partitioning problem. This heuristic includes two steps: (1) generating columns for the master problem and (2) obtaining an integer solution.

- **Generating columns for the restricted master problem**

The master problem (*set-partitioning problem*)

*minimize* $\sum_{k \in K} \sum_{r \in S_k} c_r^k z_r^k$

*subject to*

$$\sum_{k \in K} \sum_{r \in S_k} \delta_{ir}^k z_r^k = 1 \qquad \forall i \in N \qquad (10)$$

$$\sum_{r \in S_k} z_r^k = 1 \qquad \forall k \in K \qquad (11)$$

$$z_r^k = 0 \text{ or } 1 \qquad \forall k \in K, \forall r \in S_k. \qquad (12)$$

*where*: $z_r^k = 1$ if route $r \in S_k$ is selected, 0 otherwise; $\delta_{ir}^k = 1$ if job $i$ is served on route $r \in S_k$, 0 otherwise; $c_r^k$ : cost of route $k$; $S_k$ : set of routes for vehicle $k$; $K$: set of vehicles. A vehicle route starts at the depot (or at the vehicle's drop-off location in the dynamic case) visiting some nodes (each node exactly once) within their time-windows and finishes at the end depot.

The set-partitioning model selects routes covering all nodes, each node exactly once, with minimal cost. The linear relaxation of this problem (binary constraint set (12) is replaced by $z_r^k \geq 0$ ) is called the *restricted master problem*. The optimal solution of the restricted master problem is a lower bound on the objective value of the integer master problem.

To get an initial feasible solution for the restricted master problem, we introduce artificial variables $y_i \geq 0$ ($i \in N$) and modify the restricted master problem as follows (Savelsbergh and Sol, 1998):

*minimize* $\sum_{k\in K}\sum_{r\in S_k} c_r^k z_r^k + \sum_{i\in N} p y_i$ $p > \max_{k\in K, r\in S_k} c_r^k$ is a high penalty cost,

*subject to*

$$\sum_{k\in K}\sum_{r\in S_k} \delta_{ir}^k z_r^k + y_i = 1 \quad \forall i \in N \qquad (13)$$

$$\sum_{r\in S_k} z_r^k \le 1 \quad \forall k \in K \qquad (14)$$

$$z_r^k \ge 0 \quad \forall k \in K, \forall r \in S_k \qquad (15)$$

$$y_i \ge 0 \quad \forall i \in N \qquad (16)$$

An obvious feasible solution is $y_i = 1$ for all $i \in N$ and all other variables are zero.

The pricing problem (*shortest-path problem with time-windows*)

Suppose that the restricted master problem has a feasible solution $z$. Let $u_i$ ($i \in N$) be dual variables corresponding to the constraint set (10), and $v_k$ ($k \in K$) be dual variables corresponding to the constraint set (11). According to the linear programming duality (Ahuja et al., 1993), $z$ is optimal for the restricted master problem if and only if for all $k \in K$ and $r \in S_k$ the reduced cost $d_r^k$ is nonnegative, i.e. $d_r^k = c_r^k - \sum_{i\in N} \delta_{ir}^k u_i - v_k \ge 0$ for all $k \in K$ and $r \in S_k$.

The pricing problem is $\min\left\{ c_r^k - \sum_{i\in N} \delta_{ir}^k u_i - v_k \mid k \in K, r \in S_k \right\}$, in which the cost of route $r \in S_k$ is $c_r^k = \sum_{i\in N} (D_{ir} - e_i)\delta_{ir}^k$ ($D_{ir}$: the service time of node $i$ in the route $r \in S_k$). The vehicle travel distance is not present in the route cost function, however it is reflected in the service time at nodes ($D_{ir}$). Therefore this problem is a type of *shortest-path problem with time-windows* (SPPTW). If the solution of the pricing problem ($z$) results in min $d_r^k \ge 0$, $z$ is an optimal solution to the restricted master problem. We then use an interactive scheme (column-generation) to generate a set of good columns for the integer master problem. We also get a good lower bound for the integer master problem.

In this research, we have solved the SPPTW using the *generalized permanent labeling* (*GPL*) algorithm (Desrochers and Soumis, 1988) with *bucket* implementation (Dernado and Fox, 1979). In implementation, we keep track of visited nodes by a list. A route which visits a node twice (or more) will be eliminated to avoid creation of a path with cycle(s).

In many VBIT systems, there are only one-sided time-windows at pickup locations and no time-windows are required at delivery locations. In that case, we add artificial time-windows for nodes, since the *GPL* algorithm needs two-sided time-windows to perform. Adding too

long time-windows dramatically slows down the *GPL* algorithm. In contrast to this, too short time-windows may cut off the optimal solution. Generally, the *GPL* algorithm works best for cases where time-windows at nodes are tight. In cases where very wide time-windows exist at pickup locations, the running-time of the *GPL* algorithm and therefore the column-generation algorithm may increase dramatically along with the problem size.

Column-generation scheme

- *Step 0*: solve the modified restricted master problem by the simplex algorithm (CPLEX),
- *Step 1*: get dual variables ($u_i$ and $v_k$),
- *Step 2*: solve the pricing problem using the *GPL* algorithm. If the pricing problem's objective value ≥ 0, STOP. Otherwise, add newly generated column into the (modified) restricted master problem and go to Step 0.

- **Obtaining an integer solution**

We observe that when a limited number of loads (about four loads per vehicle) is taken into account for scheduling, we obtain a very good solution by solving the integer master problem with the set of columns obtained in the column-generation step. We may then improve the solution using improvement algorithms (section 4.1.2). In implementation, we replaced the set of set-partitioning constraints (13) by a set of set-covering constraints ($\sum_{k\in K}\sum_{r\in S_k}\delta_{ir}^k z_r^k + y_i \geq 1$), since we found in the experiments that using a set-covering formulation leads to better overall solutions.

Framework for column-generation heuristic

- *Step 0*: solve the restricted master problem by column-generation approach. The optimal value of this problem is a lower bound for the integer master problem.
- *Step 1*: solve the integer master problem with the columns obtained in the previous step using CPLEX,
- *Step 2*: if the objective value equals the lower bound (obtained in step 0), STOP, otherwise improve the resulting solution using improvement algorithms described in section 4.1.2. STOP.

## 4.2 Computational results for the static case

Experimental environments (different layouts used) and parameters are described in section 5.1. The three heuristics have been coded in C++. For solving the set-covering problem we

use CPLEX 7.1 from ILOG. To run all experiments we use a Toshiba Satellite Pro 2100 notebook (CPU: Mobile Intel Pentium 2GHz, 256MB ram). Input data has been generated using ten different seeds (for random numbers) corresponding to ten runs.

**Table 1** Computational results (total waiting times) for the static case (U-layout)

| **U layout** | | | *Run* | | | | | | | | | | *performance* | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *IA* | *dist* | *Alg* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *avg* | *gap%* | *RT(s)* |
| **2 vehicles, 12 loads** | | | | | | | | | | | | | | | |
| 8 | *uni* | *ins* | 34 | 73 | 137 | 103 | 95 | 176 | 59 | 98 | 41 | 173 | **98.9** | **13.7** | <1 |
| | | *com* | 34 | 73 | 128 | 103 | 95 | 164 | 51 | 63 | 41 | 173 | **92.5** | **7.7** | <1 |
| | | *col* | 34 | 69 | 117 | 83 | 95 | 156 | 51 | 59 | 30 | 163 | **85.7** | **0.4** | 1.5 |
| | | *LB* | 34 | 69 | 117 | 83 | 91.5 | 156 | 51 | 59 | 30 | 163 | **85.4** | | |
| | *exp* | *ins* | 61 | 68 | 169 | 162 | 119 | 202 | 110 | 149 | 101 | 183 | **132.4** | **20.6** | <1 |
| | | *com* | 61 | 68 | 153 | 102 | 113 | 182 | 78 | 72 | 101 | 183 | **111.3** | **5.5** | <1 |
| | | *col* | 61 | 68 | 153 | 102 | 98 | 175 | 78 | 64 | 89 | 173 | **106.1** | **0.9** | 1.2 |
| | | *LB* | 59 | 68 | 146 | 102 | 98 | 175 | 78 | 64 | 89 | 173 | **105.2** | | |
| **6 vehicles, 36 loads** | | | | | | | | | | | | | | | |
| 3 | *uni* | *ins* | 311 | 79 | 157 | 295 | 155 | 145 | 107 | 275 | 245 | 161 | **193.0** | **37.3** | <1 |
| | | *com* | 220 | 51 | 120 | 268 | 97 | 130 | 86 | 219 | 205 | 128 | **152.4** | **20.6** | <1 |
| | | *col* | 213 | 40 | 96 | 265 | 92 | 130 | 54 | 142 | 162 | 112 | **130.6** | **7.3** | 45.2 |
| | | *LB* | 204 | 40 | 93 | 215 | 68 | 128 | 54 | 141 | 160 | 108 | **121.1** | | |
| | *exp* | *ins* | 420 | 29 | 103 | 199 | 189 | 138 | 315 | 163 | 523 | 327 | **240.6** | **33.9** | <1 |
| | | *com* | 350 | 18 | 84 | 154 | 92 | 117 | 236 | 127 | 405 | 301 | **188.4** | **15.6** | <1 |
| | | *col* | 350 | 18 | 84 | 110 | 68 | 115 | 191 | 102 | 381 | 248 | **166.7** | **4.6** | 35 |
| | | *LB* | 326 | 18 | 84 | 106 | 62 | 114 | 187 | 101 | 353 | 239 | **159.0** | | |

*IA*, *dist*: load inter-arrival time mean value (time units) and distribution; *uni*, *exp*: uniform, exponential distributions; *Alg*: algorithm; *ins*, *com*, *col*: insertion, combined and column generation heuristics; *LB*: lower bound; *avg*: average of total waiting time (time units); *gap%*: gap with lower bound; *RT*: running time (CPU time - seconds).

**Table 2** Computational results (total waiting times) for the static case (I-layout)

| **I layout** | | | | | | | | | | | | | *performance* | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *IA* | *dist* | *Alg* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *avg* | *gap%* | *RT(s)* |
| **2 vehicles, 12 loads** | | | | | | | | | | | | | | | |
| 8 | *uni* | *ins* | 46 | 116 | 169 | 93 | 59 | 208 | 47 | 206 | 58 | 189 | **119.1** | **23.8** | <1 |
| | | *com* | 46 | 115 | 135 | 93 | 56 | 194 | 47 | 142 | 24 | 125 | **97.7** | **7.2** | <1 |
| | | *col* | 46 | 90 | 128 | 71 | 56 | 194 | 47 | 142 | 10 | 125 | **90.9** | **0.2** | 1.3 |
| | | *LB* | 46 | 90 | 126 | 71 | 56 | 194 | 47 | 142 | 10 | 125 | **90.7** | | |
| | *exp* | *ins* | 67 | 131 | 203 | 90 | 86 | 242 | 110 | 183 | 50 | 183 | **134.5** | **17.2** | <1 |
| | | *com* | 67 | 112 | 180 | 86 | 72 | 223 | 110 | 181 | 33 | 157 | **122.1** | **8.8** | <1 |
| | | *col* | 67 | 112 | 138 | 84 | 72 | 213 | 110 | 153 | 21 | 157 | **112.7** | **1.2** | 1.6 |
| | | *LB* | 67 | 112 | 134 | 84 | 72 | 213 | 110 | 145 | 21 | 156 | **111.4** | | |
| **6 vehicles, 36 loads** | | | | | | | | | | | | | | | |
| 3 | *uni* | *ins* | 342 | 69 | 183 | 335 | 190 | 167 | 175 | 315 | 301 | 205 | **228.2** | **44.2** | <1 |
| | | *com* | 255 | 43 | 145 | 227 | 134 | 120 | 133 | 298 | 186 | 131 | **167.2** | **23.8** | <1 |
| | | *col* | 261 | 33 | 73 | 219 | 98 | 82 | 77 | 273 | 176 | 110 | **140.2** | **9.1** | 55.9 |
| | | *LB* | 243 | 31 | 71 | 208 | 86 | 75 | 77 | 215 | 167 | 101 | **127.4** | | |
| | *exp* | *ins* | 489 | 24 | 122 | 190 | 181 | 99 | 311 | 228 | 381 | 374 | **239.9** | **32.4** | <1 |
| | | *com* | 421 | 16 | 80 | 159 | 96 | 66 | 233 | 167 | 320 | 278 | **183.6** | **11.6** | <1 |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *col* | 418 | 16 | 66 | 135 | 57 | 62 | 206 | 166 | 312 | 278 | **171.6** | **5.5** | 48.7 |
| | | *LB* | 407 | 15 | 57.5 | 108 | 56 | 62 | 201 | 140 | 306 | 270 | **162.2** | | |

Table 1 and Table 2 show that the combined heuristic gains significant improvements in comparison with the insertion heuristic without increasing running-times. The column-generation heuristic obtains better results overall (obtaining optimal solutions in many cases when 2 vehicles are used). However, when the number of vehicles increases to 15 or more, this heuristic will take a considerable amount of time (can be half an hour or more depending on the problem) to run and may not satisfy real-time scheduling requirements.

### 4.3 The real-time scheduling problem

#### 4.3.1 Dynamic scheduling using rolling horizons

In VBIT systems, we may know information about load arrivals during a time period *T* in advance. This information may be not hundred percent certain. Based on this information we propose two rolling-horizon strategies including rolling by time and rolling by the number of loads.

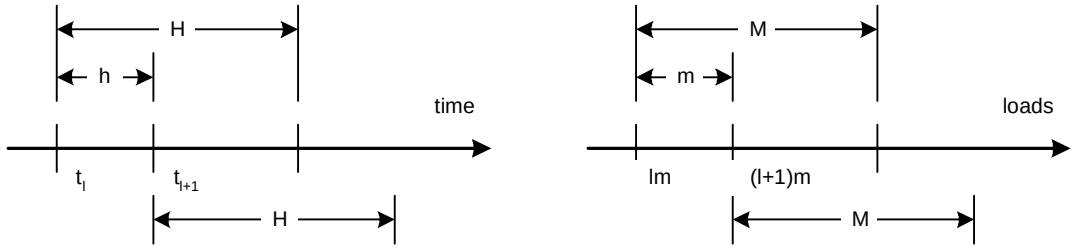



**Figure 2** Rolling horizon illustration (by time - left and by the number of loads - right)

**Rolling by time horizon** (see Figure 2)

Using this rolling horizon policy (Psaraftis, 1988), we schedule all (known) loads during a time period $H$ ($0 \le H \le T$) using the proposed heuristics (section 4.1). Depending on load arrival rates and load inter-arrival distributions during the operating period, the number of scheduled loads can differ significantly for the time horizon $H$. However, vehicles only follow the resulting schedule during a time period $h = aH$ ($a < 1$, normally 0.4 – 0.6). After the time period $h$ the system invokes the scheduling algorithm again to schedule all known loads in the period $[h, h + H]$. The process stops when all loads are transported.

**Rolling by the number of loads** (see Figure 2)

As described in the time horizon policy, the number of scheduled loads at each step can differ significantly. When too many loads are taken into account, the running time of the scheduling

algorithms may increase significantly and may not catch up with real-time events. A solution is to reduce the length of the time horizon. However, this may lead to insufficient loads available for scheduling, which limits the quality of the algorithm. Therefore, we propose a second rolling horizon policy - rolling by the number of loads. Suppose that during time period *T*, we know at least *L* loads in advance. This policy works as follows:

- Schedule *M* loads which are known in advance ($0 \le M \le L$) using the proposed heuristics,
- Re-schedule vehicles after the $m^{th}$ load ($m = \lceil a*M \rceil$, $a < 1$) has been picked up by solving the scheduling problem again for the next following *M* loads,
- Repeat this process until all loads have been transported. STOP.

With this policy, we can always monitor the running time of the scheduling algorithm and keep it at an acceptable level.

**Combined rolling horizon**

Practically, we may combine two rolling horizon policies into a combined one. When the number of loads known in advance is sufficient ($L \ge M$), we apply the rolling by the number of loads method, otherwise the time rolling horizon is used.

### 4.3.2 Dynamic scheduling using assignment algorithm

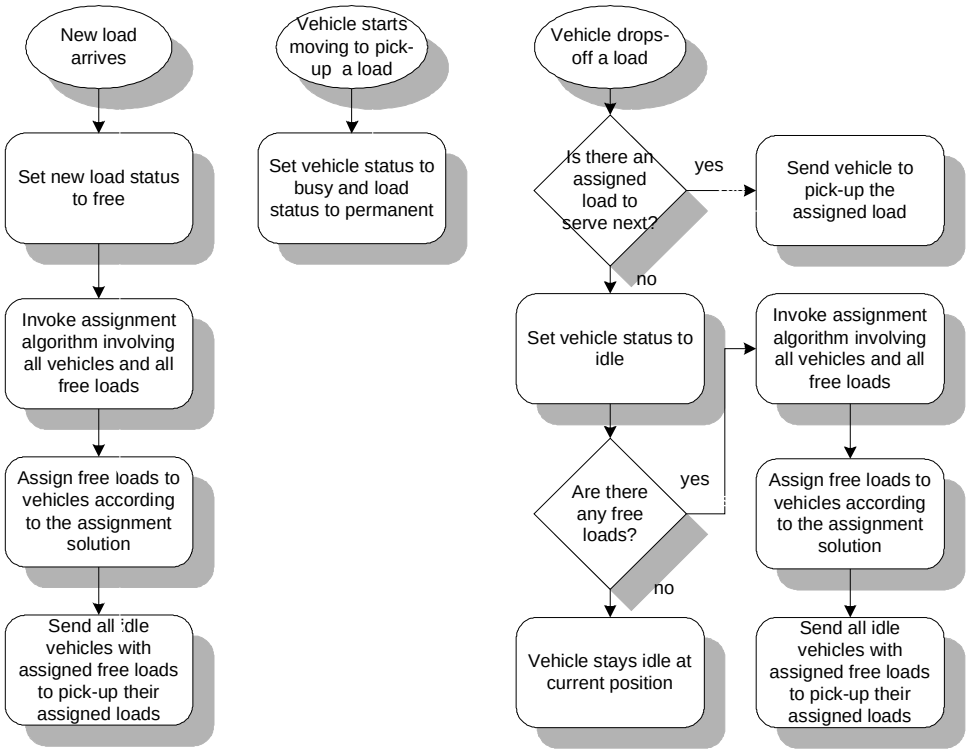
**Dynamic scheduling using assignment algorithm – a simple implementation (*DAS*)**

An intuitive scheduling approach is assigning next load to each vehicle, using an assignment algorithm. Fleischmann et al. (2004) use this approach to dynamically solve the full-truckload dispatching problem of a courier service. The main objectives in Fleischmann et al. (2004) include minimizing the order delay and the vehicle empty travel time. These are not relevant in our case, as we focus on minimizing the average load waiting time, so we adopt new cost functions in our implementation. We use the assignment algorithm of Jonker and Volgenant (1987) to solve the assignment problem. Dummy loads and dummy vehicles (as in Fleischmann et al., 2004) are introduced to balance the number of loads and vehicles for the assignment algorithm. We adapt four types of involved costs as follows:

- The cost of assigning a real vehicle to a real load ($f_{main}$) equals $C_{empt} \times Travtime$ plus $C_{wait} \times (Lwaittime)^{\alpha}$. *Travtime* is the vehicle travel time from its available location (current location for an idle vehicle and the vehicle's current load drop-off location for a busy vehicle) to a load release location and *Lwaittime* is the estimated waiting time of corresponding load.

- The cost of assigning a real vehicle to a dummy load is the unattractiveness cost of a location (vehicle waits at it current location): $C_{loc} \times 1$,
- The cost of assigning a dummy vehicle to a real load (load waits and remains unassigned at its release location) ($f_{urgency}$) equals $C_{urg}/(\textit{load release time} + \textit{time window size} - \textit{current time})^{\beta}$ if $(\textit{load release time} + \textit{time window size}) > (\textit{current time})$ and equals $\infty$ otherwise,
- The cost of assigning a dummy vehicle to a dummy load (irrelevant cost) is 0.

The values of the cost coefficients in our implementation are $C_{empt} = 10$, $C_{wait} = 2$, $C_{loc} = 5\times10^3$, $C_{urg} = 2\times10^7$, $\alpha = 2$, $\beta = 1$ or 2 (for I- and U-layout respectively – section 5). Several of the cost coefficients are taken from Fleischmann et al. (2004) ($C_{loc}$, $C_{urg}$, $\alpha$). Other good cost coefficients are obtained from experiments. In our problem, we have only one-sided time-window for loads and the cost function $f_{main}$ is in favor of loads with smaller waiting times. This may lead to a very high value of the maximum load waiting time, so we introduce an artificial time-window for loads to guarantee an acceptable value of the maximum load waiting time. The general operating framework for the scheduling approach using the dynamic algorithm is illustrated in Figure 3 (adapted from Fleischmann et al., 2004).




*Free load*: a load already arrived but not assigned to any vehicle or the assigned vehicle is still busy serving another load. A *busy vehicle* will be available at its current load drop-off location at drop-off time.

**Figure 3** The general framework for the dynamic assignment algorithm

**Look-ahead dynamic assignment algorithm (*LAS*)**

Obviously, the assignment algorithm works best for the case where we can assign about one load to each vehicle, but normally, with the implementation of Figure 3, we do not have enough loads to assign to all vehicles. In addition, we may know some information about future load arrivals, which we could use to improve *DAS*. Therefore in this section, we introduce a new real-time scheduling policy, a look-ahead dynamic assignment algorithm (*LAS*). *LAS* schedules vehicles using the same approach as *DAS*, however besides free loads the assignment algorithm take into account also loads which are known to arrive during a look-ahead period $T_L$. A good length for $T_L$ is the period during which about $K$ (the number of vehicles) loads are known to arrive ($T_L = K \times \tau$, $\tau$ is the load inter-arrival time). We can consider *LAS* a special case of the rolling by time policy in which $H$ equals $K \times \tau$ and $h$ equals *min*{time that a new load arrives, time until the first vehicle drops-off its load} from current time.

### 4.3.3 Vehicle dispatching rules

We briefly describe the two best dispatching rules in De Koster et al., 2004 here. These rules are the nearest-vehicle-first and the nearest-vehicle-first with look-ahead rules.

**Nearest-Vehicle-First (*NVF*)**

According to the *NVF* rule, when a load enters the system, it places a move request; the shortest distance along the traveling paths to every available vehicles, is then calculated. The idle vehicle, whose travel distance is the shortest, is dispatched to the point of request. When a vehicle becomes idle, it searches for the closest load.

**Nearest-Vehicle-First with look-ahead (*NVF_LA*)**

*NVF_LA* operates similarly to *NVF*. The difference is that the load gives a signal $\Delta$ time units prior to its actual release time. The time between the actual release, and the virtual release $\Delta$ time units before, can be interpreted as a look-ahead time. This gives the vehicle the opportunity to travel to the load before the load is physically ready for transport. The vehicle can therefore arrive just before or after the load is ready for transport, thereby reducing load-waiting times.

# 5 Experiments

## 5.1 Experimental setup

In this section, we describe the layout characteristics, load arrival and further data used in our experiments.

**Layouts of experimental environments**

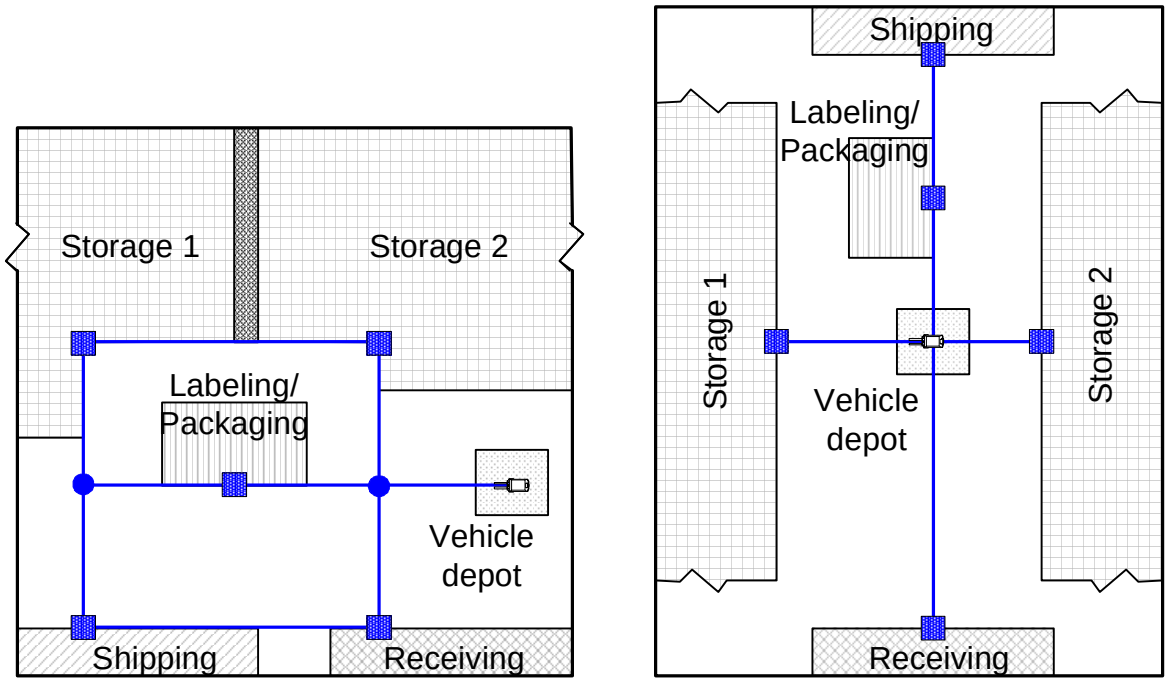



**Figure 4** Layouts of experimental environments (U left and I right)

We select two warehouse environments for experiments. Depending on locations and functions of warehouses several warehouse layout types exist (Tompkins et al., 2003). We select U- and I-layout warehouses, which are very common in practice (Tompkins et al., 2003; Van der Meer, 2000). In U-layout warehouses storage is a main function. I-layouts are used, when transshipment is an important function and it is possible for trucks to arrive at different sides of the warehouse. The travel distance matrices for both layouts are given below:

**Table 3** Distance matrices for two layouts

| U layout | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Location* | | *0* | *1* | *2* | *3* | *4* | *5* |
| *Depot* | *0* | 0 | 10 | 20 | 10 | 10 | 20 |
| *Receiving* | *1* | 10 | 0 | 20 | 10 | 10 | 10 |
| *Storage 1* | *2* | 20 | 20 | 0 | 10 | 10 | 10 |

| I layout | | | | | | | |
|---|---|---|---|---|---|---|---|
| | *0* | *1* | *2* | *3* | *4* | *5* |
| *0* | 0 | 10 | 6 | 4 | 5 | 10 |
| *1* | 10 | 0 | 16 | 14 | 15 | 20 |
| *2* | 6 | 16 | 0 | 10 | 11 | 16 |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Storage 2* | *3* | 10 | 10 | 10 | 0 | 10 | 20 | | *3* | 4 | 14 | 10 | 0 | 9 | 14 |
| *Labeling* | *4* | 10 | 10 | 10 | 10 | 0 | 10 | | *4* | 5 | 15 | 11 | 9 | 0 | 5 |
| *Shipping* | *5* | 20 | 10 | 10 | 20 | 10 | 0 | | *5* | 10 | 20 | 16 | 14 | 5 | 0 |

**Performance criteria**

In VBIT systems, the crucial performance criterion is minimizing the average load waiting time (*avg*) or maximizing the throughput given a certain number of vehicles. Other criteria such as minimizing the number of loads in critical queues might be important as well. In this study, we consider minimizing the average load waiting time as the main performance criterion, and use three other secondary criteria including the maximum load waiting time (*max_wait*), the maximum number of loads in queues (*max_inQ*) and vehicle utilization (*util%*).

**Experimental parameters**

We suppose that vehicles can park at their pick-up/ drop-off locations and vehicle loading and unloading times are negligible. Since varying the load inter-arrival time and the number of vehicles has similar effects, we vary only the load inter-arrival time. Values of parameters are selected to closely reflect practical situations.

All important experimental factors and their values are described below:

- Experimental layouts (*Lay*): 2 (U and I-layouts),
- Number of vehicles (*K*): 6 (typical number in warehouses),
- Load inter-arrival distributions (*Dist*): 2 (uniform, exponential),
- Load inter-arrival times ($\tau$): 2 levels (3, 3.6),
- Scheduling algorithms and dispatching rules:
  - Two dispatching rules (*Disp. Rules*): *NVF* and *NVF_LA*. The best length of the look-ahead period ($T_L$) is taken. This value is estimated using simulation experiments (section 5.2.3).
  - Two assignment algorithms (*Assign. Algs*): *DAS* and *LAS* ($T_L = K \times \tau$),
  - Three heuristics including insertion (*Insertion*), combined (*Com-Heur*) and column-generation (*Column-Heur*) heuristics under two rolling horizon policies: by time (*T*) and by the number of loads (*M*),
- Rolling horizon parameters:
  - Rolling by the number of loads: $M = K \times 4$, $m = K \times 2$.

- Rolling by time: $H = K \times 4 \times \tau$, $h = K \times 2 \times \tau$.

- For each combination of experimental factors, we use ten replications ($N_R = 10$).

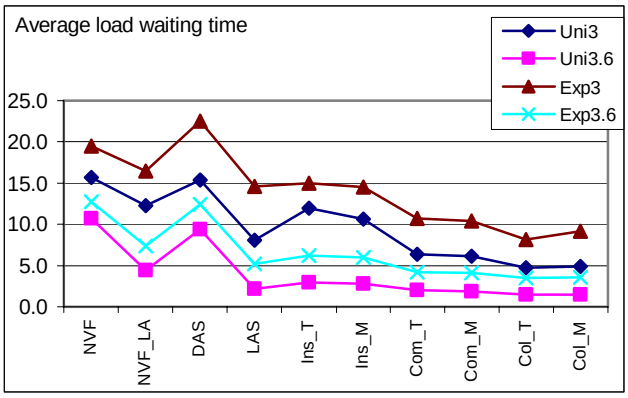
For all dynamic scheduling strategies, we set a time-window of 50 time units for all job-nodes. For *DAS* and *LAS*, we introduce a time ($T_W$) to limit the maximum load waiting time to a reasonable level. We have found that a time-window about the value of the maximum load waiting time for the corresponding condition when *NVF* is used ($T_W$ = *max_wait NVF*) is a good one.

## 5.2 Performance evaluation

### 5.2.1 Performance evaluation for the U-layout

**Table 4** Experimental results for the U-layout

| Dist | τ | perfor. measure | Disp. Rules | | Scheduling algorithms | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | *Assign. Algs* | | *Insertion* | | *Com_Heur* | | *Column_Heur* | |
| | | | *NVF* | *NVF_LA* | *DAS* | *LAS* | *T* | *M* | *T* | *M* | *T* | *M* |
| *uni* | 3 | ***avg*** | **15.70** | **12.25** | **15.36** | **8.09** | **11.96** | **10.66** | **6.33** | **6.16** | **4.74** | **4.91** |
| | | *max_wait* | 49.30 | 52.70 | 38.50 | 30.60 | 45.90 | 45.80 | 39.70 | 39.40 | 41.00 | 41.80 |
| | | *max_inQ* | 6.90 | 8.40 | 6.20 | 7.60 | 6.20 | 5.70 | 4.60 | 4.70 | 4.30 | 4.30 |
| | | *util%* | 95.99 | 92.19 | 92.65 | 98.68 | 94.74 | 94.86 | 93.08 | 93.09 | 91.23 | 92.04 |
| | 3.6 | ***avg*** | **10.74** | **4.42** | **9.42** | **2.14** | **2.96** | **2.79** | **1.99** | **1.89** | **1.49** | **1.48** |
| | | *max_wait* | 32.60 | 31.50 | 25.70 | 17.30 | 21.20 | 20.90 | 27.80 | 20.00 | 24.50 | 23.30 |
| | | *max_inQ* | 4.70 | 5.40 | 4.70 | 6.50 | 3.50 | 3.40 | 3.20 | 3.20 | 2.90 | 2.90 |
| | | *util%* | 86.65 | 86.21 | 79.22 | 96.83 | 84.25 | 84.25 | 82.63 | 82.83 | 81.91 | 81.93 |
| *exp* | 3 | ***avg*** | **19.51** | **16.48** | **22.52** | **14.58** | **14.98** | **14.55** | **10.70** | **10.37** | **8.17** | **9.14** |
| | | *max_wait* | 68.20 | 68.70 | 53.00 | 43.70 | 47.40 | 48.70 | 46.90 | 46.30 | 47.40 | 46.90 |
| | | *max_inQ* | 8.70 | 9.70 | 8.10 | 9.40 | 7.30 | 7.60 | 6.50 | 6.20 | 5.90 | 6.10 |
| | | *util%* | 93.81 | 91.24 | 91.69 | 97.33 | 93.27 | 93.28 | 91.57 | 91.52 | 86.83 | 90.84 |
| | 3.6 | ***avg*** | **12.72** | **7.34** | **12.39** | **5.20** | **6.18** | **5.97** | **4.17** | **4.12** | **3.46** | **3.57** |
| | | *max_wait* | 43.50 | 46.80 | 35.90 | 27.40 | 37.50 | 36.40 | 37.60 | 34.80 | 35.90 | 37.80 |
| | | *max_inQ* | 6.10 | 6.70 | 5.80 | 7.70 | 5.00 | 4.70 | 4.30 | 4.40 | 4.40 | 4.30 |
| | | *util%* | 83.18 | 82.55 | 78.75 | 94.44 | 82.84 | 83.03 | 81.26 | 80.92 | 78.70 | 80.32 |



*Uni3* (*Exp3*): the load inter-arrival distribution is uniform (exponential) distribution and the load inter-arrival time is 3 (time units); *Ins, Com, Col_T,_M*: insertion, combined and column-generation heuristics under two rolling horizon policies.

**Figure 5** Average waiting times – U-layout

Table 4 and Figure 5 indicate clearly that the average load waiting time reduces dramatically when we schedule vehicles using dynamic scheduling strategies. Best results have been obtained when we apply the column-generation heuristic to solve static instances of real-time scheduling problems. The average improvement of *Com_Heur* over *NVF* is 86.24% (uniform distribution, $\tau = 3.6$). The average reduction when we compare the performance of *NVF* and *NVF_LA* is 59.62%. In order to rank the different scheduling policies, we used a Tukey test with 95% confidence intervals. For all inter-arrival distributions tested, results can be found in Table 5. Since the two rolling horizon policies (by *T* and *M*) perform quite similarly (Table 4 and also by Tukey test), we use only one entry to represent both of them in Table 5. For example, the entry "column generation" represents both rolling horizon policies (by *T* and *M*) using column-generation heuristic. The *NVF_LA* and *LAS* perform significantly better than *NVF* and *DAS* (Table 5). Dynamic scheduling strategies are also favorable to dispatching rules considering the maximum number of load in queues and the maximum load waiting time.

**Table 5** Ranking of different scheduling policies for the U-layout (Tukey test with 95 % confidence interval)

| ***Dist*** | *Uniform* | | | | | | | *Exponential* | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\tau$ | 3 | | | 3.6 | | | | 3 | | 3.6 | |
| *Column generation* | 1 | | | 1 | | | | 1 | | 1 | |
| *Combined heuristic* | 1 | | | 2 | | | | 1 | | 2 | |
| *LAS* | 3 | | | 2 | | | | 3 | | 2 | |
| *Insertion* | 3 | | | 4 | | | | 3 | | 2 | |
| *NVF_ LA* | 3 | | | 5 | | | | 3 | | 2 | |
| *DAS* | 6 | | | 6 | | | | 6 | | 6 | |
| *NVF* | 6 | | | 7 | | | | 6 | | 6 | |

*Scheduling approaches are ranked from high to low according to the average load waiting time. The average load waiting times of scheduling approaches in the same number block are not significantly different.*

*DAS* performs a bit better than *NVF* in general but it is not significant (Table 4, Table 5). *LAS* performs very well and is about equally good as *Com_Heur* (Table 5), particularly in the high load inter-arrival time cases ($\tau = 3.6$). The scheduling strategies using combined and column-generation heuristics perform much better than the ones using the insertion heuristic (largest improvement = 42.21%). We also notice that the column-generation heuristic performs better than the combined heuristic. However for large real problems, the running-time of the

column-generation heuristic grows rapidly, so it is only suitable for small and medium cases (less than 15 vehicles).

### 5.2.2 Performance evaluation for the I-layout

**Table 6** Experimental results for the I-layout

| Dist | τ | perfor. measure | Disp. Rules | | Scheduling algorithms | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | *Assign. Algs* | | *Insertion* | | *Com_Heur* | | *Column_Heur* | |
| | | | *NVF* | *NVF_LA* | *DAS* | *LAS* | *T* | *M* | *T* | *M* | *T* | *M* |
| *uni* | 3 | ***avg*** | **40.10** | **36.11** | **27.71** | **17.73** | **19.20** | **18.47** | **12.80** | **12.45** | **10.57** | **10.40** |
| | | *max_wait* | 204.20 | 189.40 | 59.30 | 49.10 | 49.30 | 49.30 | 49.20 | 49.50 | 49.20 | 49.50 |
| | | *max_inQ* | 19.20 | 17.90 | 8.70 | 9.80 | 8.10 | 7.80 | 6.90 | 6.70 | 6.40 | 6.70 |
| | | *util%* | 96.74 | 96.43 | 94.89 | 97.94 | 95.98 | 96.05 | 95.69 | 95.65 | 93.73 | 95.02 |
| | 3.6 | ***avg*** | **14.73** | **10.64** | **13.27** | **3.29** | **4.87** | **4.91** | **3.04** | **3.04** | **2.46** | **2.46** |
| | | *max_wait* | 66.50 | 70.10 | 34.00 | 22.00 | 32.50 | 28.80 | 35.00 | 33.00 | 34.40 | 33.40 |
| | | *max_inQ* | 6.90 | 7.80 | 5.50 | 6.80 | 4.30 | 4.40 | 3.90 | 3.70 | 3.50 | 3.80 |
| | | *util%* | 89.05 | 87.98 | 82.61 | 95.24 | 86.40 | 86.23 | 84.95 | 85.25 | 84.45 | 84.56 |
| *exp* | 3 | ***avg*** | **44.19** | **42.25** | **34.76** | **25.42** | **19.45** | **18.73** | **14.14** | **14.40** | **13.81** | **12.66** |
| | | *max_wait* | 214.00 | 213.50 | 74.40 | 66.10 | 50.00 | 49.80 | 49.50 | 49.70 | 51.70 | 48.60 |
| | | *max_inQ* | 20.80 | 20.30 | 10.20 | 11.20 | 8.00 | 8.00 | 7.10 | 7.70 | 7.20 | 7.10 |
| | | *util%* | 95.89 | 95.68 | 93.48 | 96.89 | 94.31 | 93.93 | 94.04 | 94.06 | 93.57 | 93.51 |
| | 3.6 | ***avg*** | **18.73** | **16.05** | **17.02** | **7.33** | **8.74** | **8.57** | **6.07** | **6.08** | **5.50** | **5.55** |
| | | *max_wait* | 93.90 | 91.10 | 48.70 | 38.40 | 43.50 | 43.50 | 44.50 | 44.50 | 43.30 | 43.40 |
| | | *max_inQ* | 9.80 | 9.60 | 6.70 | 8.00 | 6.10 | 6.00 | 5.10 | 5.40 | 4.90 | 5.10 |
| | | *util%* | 87.03 | 86.73 | 81.74 | 93.40 | 85.32 | 84.73 | 83.50 | 83.81 | 83.10 | 83.29 |

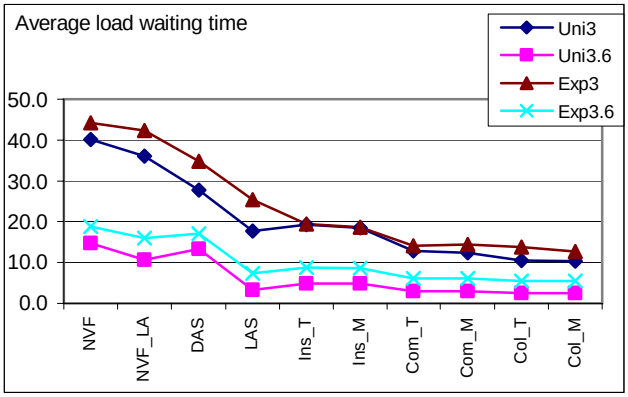



**Figure 6** Average waiting times – I-layout

We observe similar effects of using different dynamic scheduling and dispatching strategies for I-layout. However, in this layout improvements are smaller than in the U-layout. The average improvement of *Com_Heur* over *NVF* is 83.3% (uniform distribution, $\tau$ = 3.6). In this layout, the look-ahead dispatching rule (*NVF_LA)* performs less impressive than in U-layout. The average improvement compared with *NVF* is 27.74%. We also observe that for both layouts bigger improvements are obtained for smaller load arrival rates (or higher load

inter-arrival time), corresponding to smaller vehicle utilization rates. This is similar to the findings of Yang et al. (2004) and fairly obvious since in highly utilized systems there is little chance for prematurely sending vehicles to pick-up locations. Table 6 and Table 7 show that *DAS* performs better than *NVF* but it is not significant. *LAS* instead performs more impressive (in the top group in half of the cases).

**Table 7** Ranking of different scheduling policies for the I-layout (Tukey test with 95 % confidence interval)

| ***Dist*** | *Uniform* | | | | *Exponential* | | | |
|---|---|---|---|---|---|---|---|---|
| $\tau$ | 3 | | 3.6 | | 3 | | 3.6 | |
| *Column generation* | 1 | | 1 | | 1 | | 1 | |
| *Combined heuristic* | 1 | | 1 | | | 2 | 1 | |
| *LAS* | | 3 | 1 | | | 2 | 1 | |
| *Insertion* | | 3 | 1 | | | 2 | 1 | |
| *NVF_ LA* | | 5 | | 5 | | 5 | | 5 |
| *DAS* | | 6 | | 6 | | 5 | | 5 |
| *NVF* | | 6 | | 6 | | 5 | | 5 |

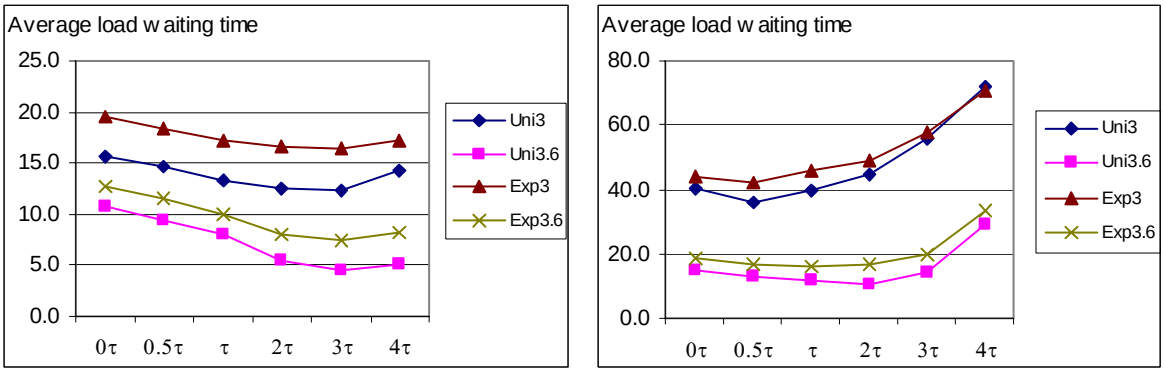
Table 7 clearly indicates that the three dynamic scheduling heuristics and *LAS* perform significantly better than dispatching rules and the simple dynamic assignment algorithm (*DAS*). Pre-arrival information has important positive influence on performance of *NVF*.

For both layouts, scheduling and dispatching strategies perform better when the load inter-arrival distribution is uniform instead of exponential. An explanation is that for uniform and exponential distributions with the same inter-arrival time, the exponential distribution has a three times higher variance. Another observation is that the average load waiting time in the U-layout is smaller than the corresponding value in the I-layout. It is caused by the average travel distance between any two points that have transportation requirements, which is higher for the I-layout than for the U-layout (13 versus 12). It is also the reason why *DAS* performs better for the I-layout. This algorithm helps to save more unnecessary movements of vehicles. Considering other performance criteria (max load waiting time, max number of loads in queues, vehicle utilization), we also find that scheduling algorithms perform better than vehicle dispatching rules. Comparing *LAS* with other scheduling approaches using rolling horizons, *LAS* performs worse in terms of the maximum number of loads in queues. *LAS* also results in a very high value of vehicle utilization. A possible reason for high utilization of *LAS* is that *LAS* is still a more local policy and according to this policy vehicles may travel longer distances.

In the next two sections, we carry out some experiments with look-ahead periods and rolling horizon policies to see how they affect the performance of dispatching rules and scheduling algorithms performance.

### 5.2.3 Influences of look-ahead periods and of rolling horizon lengths

*Influences of look-ahead periods*




*Uni3* (*Exp3*): the load inter-arrival distribution is uniform (exponential) and the load inter-arrival time is 3 (time units).

**Figure 7** Effects of the look-head period on the average load waiting time (U-layout: left, I-layout: right) for the *NVF_LA* rule.

In this research, we have experimented with 6 vehicles, 2 distributions (uniform and exponential), 2 load inter-arrival levels and have observed the influences of the look-ahead period length on two layouts and two load inter-arrival levels. We prefer to express the length of the look-ahead period in terms of the load inter-arrival time. In Figure 7 (left – U-layout), the best value for the look-ahead period for *NVF_LA* is similar for both distributions and load inter-arrival levels; it is about three times the average load inter-arrival time.

Figure 7 (right – I-layout) shows different effects. For the larger load inter-arrival time (3.6), the best value for the look-ahead period is about two times the load inter-arrival time (7.2). For smaller load inter-arrival time (3), the best look-ahead time equals half the load inter-arrival time (1.5).

Different behaviors of the look-ahead period in two layouts under different operating conditions do not permit us to recommend a specific value for the best length of the look-ahead period. Good values can only be obtained by experiments. However, the experiments indicate that it can be fairly small.

*Influences of rolling horizon lengths*

The performance of two rolling horizon policies is very similar for both layouts under various conditions, so we have experimented with only the rolling by the number of loads policy. There are six sets of rolling horizon parameters (*M*, *m*): set 1 (12, 6); set 2 (12, 8); set 3 (24, 10); set 4 (24, 12); set 5 (36, 12); set 6 (36, 18). Since all three dynamic scheduling heuristics behave similarly, we selected only the combined heuristic for experiments.

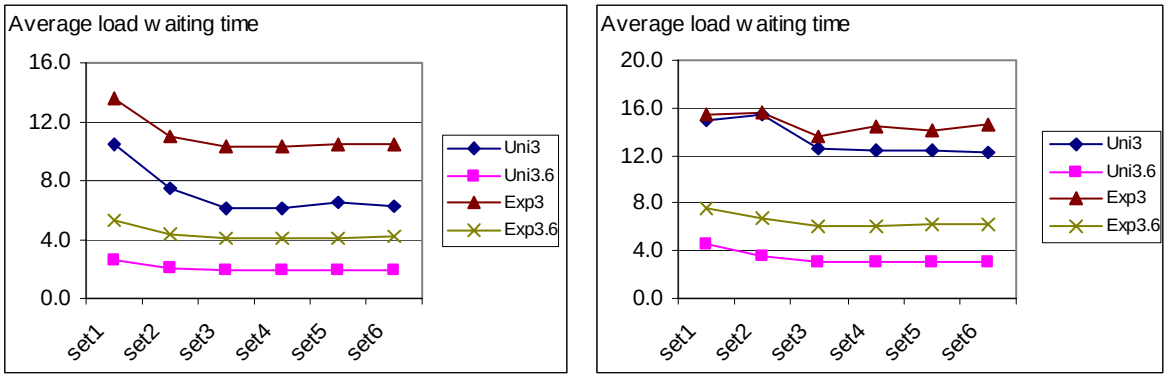



**Figure 8** Effects of rolling horizon policies using the combined heuristic on the average load waiting time (U-layout: left, I-layout: right)

Using set 1 (12, 6), we schedule about 2 loads in advance for each vehicle and let each vehicle execute about one load before re-scheduling. Figure 8 shows that significant improvements start when we schedule vehicles using set 3 (24, 10). With this set, we schedule about four loads per vehicle and let each vehicle execute about 1.7 loads. Taking more information into account (set 5, set 6), we cannot obtain a further significant improvement. However, we are not interested in scheduling vehicles too far in advance. In general, we gain significant improvements when we schedule more than about three loads per vehicle and each vehicle should transport about two loads before re-scheduling.

## 6 Conclusions and further research

In this research, we have studied the real-time scheduling problem for vehicle-based internal transport systems. We have formulated this problem as *m*-TSPTW and have proposed two rolling horizon approaches for solving it. We have proposed two new heuristics (combined and column-generation) to solve static instances of the real-time dynamic scheduling problem. We have also proposed another good dynamic scheduling strategy - *LAS*. The proposed dynamic scheduling strategies performance has been compared with the performance of two of the best vehicle dispatching rules which are known from literature.

We have found that dynamic scheduling strategies consistently outperform vehicle dispatching rules in two experimental environments and under different operating conditions. Improvements are remarkable when applying combined and column-generation to solve static instances. However, significant improvements are only possible when we know sufficient information about future loads. We have also shown that the new look-ahead dynamic algorithm (*LAS*) performs significantly better than dispatching rules and a simple dynamic assignment algorithm. The main disadvantage of *LAS* is that *LAS'* performance depends on the cost functions and the cost parameters. Depending on applications we have to select the right cost functions and tune cost parameters carefully.

We did not explicitly investigate the combined rolling horizon policy. This may provide a better result than applying a single rolling horizon policy in systems with high variation of load arrivals. The performance of the column-generation heuristic can also be improved. These may be topics for future research.

## References


Ahuja, K., Magnanti, T.L. and Orlin, J.B., Network flows: Theory, algorithms, and applications, Prentice Hall, Inc. (1993).

Cordeau, J.-F., Gendreau, M., Laporte, G., Potvin, J.-Y. and Semet, F., "A guide to vehicle routing heuristics", *Journal of the Operational Research Society*, **53**, 512-522 (2002).

De Koster, R., Le Anh, T. and Van der Meer, J.R., "Testing and classifying vehicle dispatching rules in three real-world settings", *Journal of Operations Management*, **22**(4), 369-386 (2004).

Dernado, E.V. and Fox, B.L., "Shortest-route methods: reaching, pruning, and buckets", *Operations Research*, **27**(1), 161-186 (1979).

Desaulniers, G., Lavigne, J. and Soumis, F., "Multi-depot vehicle scheduling problems with time windows and waiting costs", *European Journal of Operational Research*, **111**, 479-494 (1998).

Desrochers, J., Lenstra, J.K., Savelsbergh, M.W.P. and Soumis, F., Vehicle routing with time windows: optimization and approximation, in *Vehicle Routing: Methods and Studies*, eds. B.L.Golden and A.A.Assad, Elsevier Science Publishers, pp. 65-84 (1988).

Desrochers, M. and Soumis, F., "A generalized permanent labeling algorithm for the shortest path problem with time windows", *INFOR*, **26**(3), 191-212 (1988).

Dumas, Y., Desrosiers, J. and Soumis, F., "The pickup and delivery problem with time windows", *European Journal of Operational Research*, **54**, 7-22 (1991).

Egbelu, P.J. and Tanchoco, J.M.A., "Characterization of automated guided vehicle dispatching rules", *International Journal of Production Research*, **22**(3), 359-374 (1984).

Fleischmann, B., Gnutzmann, S. and Sandvoß, E., "Dynamic vehicle routing based on on-line traffic information", *Transportation Science*, (2004, to apear).

Jonker, R. and Volgenant, T., "A shortest augmenting path algorithm for dense and sparse linear assignment

problems", *Computing*, **38**, 325-340 (1987).

Kindervater, G.A.P. and Savelsbergh, M.W.P., "Local search in physical distribution management", *Report EUR-CS-92-05,* (1992).

Laporte, G., Gendreau, M., Potvin, J.-Y. and Semet, F., "Classical and modern heuristics for the vehicle routing problem", *Intl. Trans. in Op. Res.*, **7**, 285-300 (2000).

Le-Anh, T. and De Koster, R., "A review of design and control of automated guided vehicle systems", *Technical Report ERS-2004-031-LIS, Erasmus Research Institute of Management (ERIM), Erasmus University Rotterdam,* (2004).

Psaraftis, H.N., Dynamic vehicle routing problems, in *Vehicle Routing: Methods and Studies*, ed B.L.Golden and A.A.Assad (Eds.), Elsevier Science Publishers, pp. 233-248 (1988).

Savelsbergh, M.W.P. and Sol, M., "The general pickup and delivery problem", *Transportation Science,* **29**(1), 17-29 (1995).

Savelsbergh, M. and Sol, M., "DRIVE: Dynamic routing of independent vehicles", *Operations Research,* **46**(4), 474-490 (1998).

Tompkins, J.A., White, J.A., Bozer, Y.A. and Tanchoco, J.M.A., Facilities planning, 3rd edn, John Wiley & Sons, Inc. (2003).

Van der Meer, J.R., Operational control of internal transport system, Ph.D. Thesis, Erasmus University Rotterdam (2000).

Xu, H., Chen, Z.-L., Rajagopal, S. and Arunapuram, S., "Solving a practical pickup and delivery problem", *Transportation Science,* **37**(3), 347-364 (2003).

Yang, J., Jaillet, P. and Mahmassani H., "Real-Time multi-vehicle truckload pickup and delivery problems", *Transportation Science,* **38**(2), 135-148 (2004).

# Publications in the Report Series Research* in Management

## ERIM Research Program: "Business Processes, Logistics and Information Systems"

2004

*Smart Pricing: Linking Pricing Decisions with Operational Insights*
Moritz Fleischmann, Joseph M. Hall and David F. Pyke
ERS-2004-001-LIS
http://hdl.handle.net/1765/1114

*Mobile operators as banks or vice-versa? and: the challenges of Mobile channels for banks*
L-F Pau
ERS-2004-015-LIS
http://hdl.handle.net/1765/1163

*Simulation-based solution of stochastic mathematical programs with complementarity constraints: Sample-path analysis*
S. Ilker Birbil, Gül Gürkan and Ovidiu Listeş
ERS-2004-016-LIS
http://hdl.handle.net/1765/1164

*Combining economic and social goals in the design of production systems by using ergonomics standards*
Jan Dul, Henk de Vries, Sandra Verschoof, Wietske Eveleens and Albert Feilzer
ERS-2004-020-LIS
http://hdl.handle.net/1765/1200

*Factory Gate Pricing: An Analysis of the Dutch Retail Distribution*
H.M. le Blanc, F. Cruijssen, H.A. Fleuren, M.B.M. de Koster
ERS-2004-023-LIS
http://hdl.handle.net/1765/1443

*A Review Of Design And Control Of Automated Guided Vehicle Systems*
Tuan Le-Anh and M.B.M. De Koster
ERS-2004-030-LIS
http://hdl.handle.net/1765/1323

*Online Dispatching Rules For Vehicle-Based Internal Transport Systems*
Tuan Le-Anh and M.B.M. De Koster
ERS-2004-031-LIS
http://hdl.handle.net/1765/1324

*Generalized Fractional Programming With User Interaction*
S.I. Birbil, J.B.G. Frenk and S. Zhang
ERS-2004-033-LIS
http://hdl.handle.net/1765/1325

---

* A complete overview of the ERIM Report Series Research in Management:
https://ep.eur.nl/handle/1765/1

ERIM Research Programs:
LIS Business Processes, Logistics and Information Systems
ORG Organizing for Performance
MKT Marketing
F&A Finance and Accounting
STR Strategy and Entrepreneurship

*Meta-heuristics for dynamic lot sizing: A review and comparison of solution approaches*
Raf Jans and Zeger Degraeve
ERS-2004-042-LIS
http://hdl.handle.net/1765/1336

*Reinventing Crew Scheduling At Netherlands Railways*
Erwin Abbink, Matteo Fischetti, Leo Kroon, Gerrit Timmer And Michiel Vromans
ERS-2004-046-LIS
http://hdl.handle.net/1765/1427

*Intense Collaboration In Globally Distributed Teams: Evolving Patterns Of Dependencies And Coordination*
Kuldeep Kumar, Paul C. van Fenema and Mary Ann Von Glinow
ERS-2004-052-LIS
http://hdl.handle.net/1765/1446

*The Value Of Information In Reverse Logistics*
Michael E. Ketzenberg, Erwin van der Laan and Ruud H. Teunter
ERS-2004-053-LIS
http://hdl.handle.net/1765/1447

*Cargo Revenue Management: Bid-Prices For A 0-1 Multi Knapsack Problem*
Kevin Pak and Rommert Dekker
ERS-2004-055-LIS
http://hdl.handle.net/1765/1449

*Real-Time Scheduling Approaches For Vehicle-Based Internal Transport Systems*
Tuan Le-Anh and M.B.M. De Koster
ERS-2004-056-LIS

*A Grounded Theory Analysis Of E-Collaboration Effects For Distributed Project Management*
S. Qureshi, M. Liu and D. Vogel
ERS-2004-059-LIS
http://hdl.handle.net/1765/1448

*A Phenomenological Exploration Of Adaptation In A Polycontextual Work Environment*
P.C. van Fenema and S. Qureshi
ERS-2004-061-LIS

*Satisfaction Attainment Theory As A Model For Value Creation*
R.O. Briggs, S. Qureshi and B. Reining
ERS-2004-062-LIS
http://hdl.handle.net/1765/1450