Vehicle Routing with Cross-Docking

Min Wen[1], Jesper Larsen[1], Jens Clausen[1], Jean-François Cordeau[2,*], Gilbert Laporte[3]

[1]*Department of Informatics and Mathematical Modeling, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark*

[2]*Canada Research Chair in Logistics and Transportation, HEC Montréal, 3000 chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7*

[3]*Canada Research Chair in Distribution Management, HEC Montréal, 3000 chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7*

September 4, 2007

Abstract

Over the past decade, cross-docking has emerged as an important material handling technology in transportation. A variation of the well-known *Vehicle Routing Problem* (VRP), the *Vehicle Routing Problem with Cross-Docking* (VRPCD) arises in a number of logistics planning contexts. This paper addresses the VRPCD, where a set of homogeneous vehicles are used to transport products from the suppliers to the corresponding customers via a cross-dock. The products can be consolidated at the cross-dock but cannot be stored for very long because the cross-dock does not have long-term inventory-holding capabilities. The objective of the VRPCD is to minimize the total traveled distance while respecting time window constraints at the nodes and a time horizon for the whole transportation operation. In this paper, a mixed integer programming formulation for the VRPCD is proposed. A tabu search heuristic is embedded within an adaptive memory procedure to solve the problem. The proposed algorithm is implemented and tested on data sets provided by the Danish consultancy Transvision, and involving up to 200 pairs of nodes. Experimental results show that this algorithm can produce high quality solutions (less than 5% away from optimal solution values) within very short computational time.

Keywords: Cross-docking, Vehicle Routing Problem, Pickup and Delivery.

[*]Corresponding author (jean-francois.cordeau@hec.ca)

Introduction

Cross-docking is a relatively new warehousing strategy in logistics. It is defined as the consolidation of products from incoming shipments so that they can be easily sorted at a distribution center for outgoing shipments. The distribution center in this case is referred to as a cross-dock. It essentially eliminates the inventory holding function of a traditional warehouse while still allowing consolidation. The shipments arriving from disparate sources are regrouped and dispatched directly by the outgoing trailers without being stored. Shipments typically spend less than 24 hours at the cross-dock, sometimes less than an hour. This way, cross-docking not only provides good customer service but also yields substantial advantages over traditional warehousing: reduction in inventory investment, storage space, handling cost and order-cycle time, as well as faster inventory turnover and accelerated cash flow (Cook, Gibson and MacCurdy, 2005; Apte and Viswanathan, 2000).

Due to its remarkable benefits, cross-docking has been widely adopted in practice by manufacturing and retailing companies. A successful application of cross-docking is found at Wal-Mart, the largest and highest profit retailer in the world. In the system used by this company, products are continuously delivered to Wal-Mart's cross-docks, where they are selected, repacked, and then dispatched to stores, often without ever sitting in inventory. By avoiding spending valuable time and handling inventory cost, cross-docking has enabled Wal-Mart to adopt an everyday low price strategy and has helped the company improve its market share and profitability (Stalk, Evans and Shulman, 1992). CompUSA is another major user of cross-docking: 70% of its products in dollar volume goes through a cross-dock (Gentry, 2005).

Considerable research on cross-docking has been carried out in recent years. However, most of the papers have investigated the physical design of the cross-dock (Ratliff, Vate and Zhang, 1999; Bartholdi III and Gue, 2004) and its location (Gumus and Bookbinder, 2004). The very few papers that deal with the transportation problems associated with cross-docking have studied two types of network models.

The first type of model is characterized by an "open" network, in which the distribution flow starts from a single supplier and ends at a single customer via a cross-dock without forming any loop. Work in this area includes Sung and Song (2003), Jayaraman and Ross (2003) and Chen *et al.* (2006). Sung and Song (2003) have discussed the problem of deciding whether to open a cross-dock or not, and the problem of assigning vehicles for transportation from a supplier to a single destination via one of the open cross-docks. They have proposed a tabu search algorithm for the transportation problem. Jayaraman and Ross (2003) have investigated a similar problem. Given a cost for opening each supplier, they have discussed how to decide whether a supplier should be opened or closed. Simulated annealing methods were used in their paper. In Chen *et al.* (2006), time windows for suppliers and customers are given, and the inventory cost at the cross-dock is also taken into consideration. These authors have proposed a hybrid metaheuristic combining simulated annealing and tabu search.

In the second type of network model, each vehicle leaves the cross-dock to pick up or deliver goods and returns to the cross-dock after completing its tour. To the best of our knowledge, only one publication, that of Lee, Jung and Lee (2006), has studied a transportation problem of this type. This problem consists of a single cross-dock, multiple suppliers and multiple customers. The task is to assign tours to a set of vehicles at the cross-dock so that suppliers and customers are visited within their time windows. The authors assume that all vehicles should arrive simultaneously at the cross-dock from their pickup routes. A mixed integer programming formulation and a tabu search algorithm were proposed.

The problem considered in our study is the *Vehicle Routing Problem with Cross-Docking* (VRPCD). The problem is similar to that of Lee, Jung and Lee (2006) where the vehicles can pick up or deliver more than one supplier or customer, and the pickup and delivery routes start and end at the cross-dock. However, there is no constraint on simultaneous arrival for all the vehicles in our problem. Instead, the dependency among the vehicles is determined by the consolidation decisions. Moreover, each pickup and delivery has predetermined time windows.

The remainder of this paper is organized as follows. A detailed description of the VRPCD is given in the next section. A mixed integer formulation of the problem is then presented, followed by a heuristic embedding tabu search within an adaptive memory procedure. The algorithm is implemented and tested on realistic data involving up to 200 supplier-customer pairs. Computational results are presented and conclusions follow.

Problem Definition

The VRPCD considered in this paper is in essence a problem of transporting products from a set of suppliers to their corresponding customers using a cross-docking strategy. Products from the suppliers are picked up by a fleet of homogeneous vehicles, consolidated at the cross-dock, and immediately delivered to customers by the same set of vehicles, without intermediate storage. Therefore, the problem involves not only vehicle route design, but also a consolidation decision at the cross-dock.

A small VRPCD instance of five supplier-customer pairs (requests) is shown in Figure 1. The set of nodes $\{1, ..., 5\}$ represents the suppliers and $\{1', ..., 5'\}$ represents the corresponding customers. Figure 2 illustrates the pickup and delivery routes for the three vehicles, all of which start and end their routes at the cross-dock. For example, the first vehicle makes pickups at nodes 1 and 2 and delivers to nodes $1'$, $2'$ and $5'$. Note that in the VRPCD, a supplier and its corresponding customer are not necessarily served by the same vehicle. For instance, request 5 is picked up by the third vehicle but delivered by the first vehicle. Hence, the third vehicle has to unload product 5 after it arrives at the cross-dock from its pickup route, and the first vehicle needs to load product 5 before

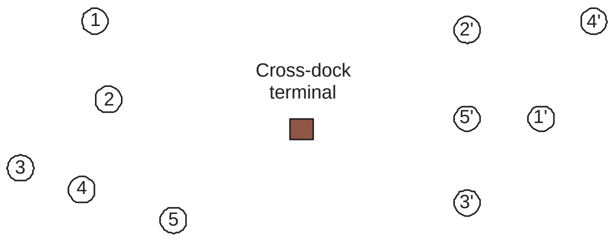

Figure 1: A small instance of the VRPCD

it leaves for its delivery route. Figure 3 shows the details of the consolidation process taking place at the cross-dock.

As in the *Vehicle Routing Problem with Time Windows* (VRPTW), each node must be served by exactly one vehicle within its time window, the accumulated load of each route must not exceed the vehicle capacity, and the time horizon for the whole transportation operation must be respected. At the cross-dock, for each vehicle the unloading must be completed before reloading starts. Each vehicle can start unloading immediately after it arrives at the cross-dock from its pickup route. The duration of the unloading consists of a fixed time for preparation, and the time needed for unloading products, equal to the time for unloading each pallet multiplied by the number of pallets. For instance, suppose vehicle k reaches the cross-dock at 10:00. It needs to unload products i_1 and i_2, whose demands are 5 and 9, respectively. Given that the fixed time for preparation is ten minutes and the time for unloading each pallet is one minute, the total unloading duration is 24 (= 10 + 5 + 9) minutes. Note that the time at which vehicle k finishes unloading, i.e. 10:24, is also the time at which products i_1 and i_2 are ready to be reloaded in their corresponding delivery vehicles.

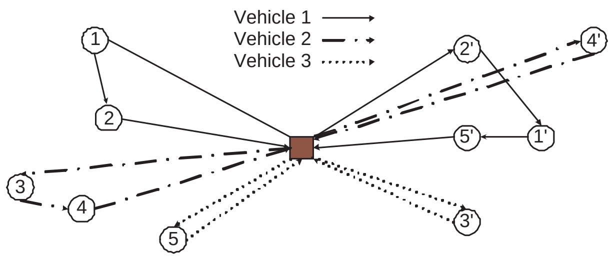

Figure 2: Vehicle routes

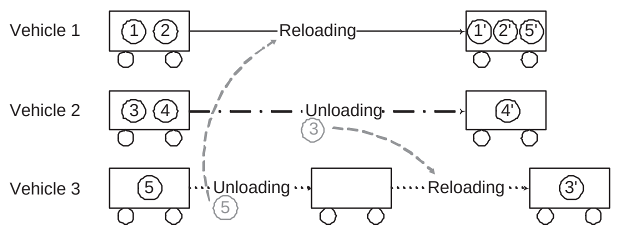

Figure 3: The consolidation process at the cross-dock

Mathematical formulation

We now present a mixed integer linear programming formulation for the VRPCD. Denote the set of pickup nodes by $P = \{1, ..., n\}$ and the set of delivery nodes by $D = \{n + 1, ..., 2n\}$. Each request i is identified by the node pair (i, $i + n$), where i is the pickup node and $i + n$ is the associated delivery node. The cross-dock is represented by four nodes and denoted by the set $O = \{o_1, o_2, o_3, o_4\}$, where the first two nodes represent the starting and ending points for pickup routes, and the last two for the delivery routes. Further, define $N = P \cup O \cup D$. The set E denotes all the feasible arcs in the network. It consists of the arcs $\{(i, j) : i, j \in P \cup \{o_1, o_2\}, i \neq j\}$ and the arcs $\{(i, j) : i, j \in D \cup \{o_3, o_4\}, i \neq j\}$. Let K be the set of vehicles.

The parameters are denoted as follows:

c_{ij} = the travel time between node i and node j $((i, j) \in E)$;
$[a_i, b_i]$ = the time window for node i $(i \in N)$;
d_i = the amount of demand of request i $(i \in P)$;
Q = the vehicle capacity;
A = the fixed time for unloading and reloading at the cross-dock;
B = the time for unloading and reloading a pallet.

The variables are:

$$x_{ij}^k = \begin{cases} 1 & \text{if vehicle } k \text{ travels from node } i \text{ to node } j \ ((i,j) \in E, k \in K) \\ 0 & \text{otherwise;} \end{cases}$$

$$u_i^k = \begin{cases} 1 & \text{if vehicle } k \text{ unloads request } i \text{ at the cross-dock } (i \in P, k \in K) \\ 0 & \text{otherwise;} \end{cases}$$

$$r_i^k = \begin{cases} 1 & \text{if vehicle } k \text{ reloads request } i \text{ at the cross-dock } (i \in P, k \in K) \\ 0 & \text{otherwise;} \end{cases}$$

$$g_k = \begin{cases} 1 & \text{if vehicle } k \text{ has to unload at the cross-dock } (k \in K) \\ 0 & \text{otherwise;} \end{cases}$$

$$h_k = \begin{cases} 1 & \text{if vehicle } k \text{ has to reload at the cross-dock } (k \in K) \\ 0 & \text{otherwise;} \end{cases}$$

s_i^k = the time at which vehicle k leaves node i $(i \in N, k \in K)$;

t_k = the time at which vehicle k finishes unloading at the cross-dock $(k \in K)$;

w_k = the time at which vehicle k starts reloading at the cross-dock $(k \in K)$;

v_i = the time at which request i is unloaded by its pickup vehicle at the cross-dock $(i \in P)$.

In addition, M is an arbitrarily large constant.

The VRPCD can be formulated as follows:

$$\text{minimize} \quad \sum_{(i,j)\in E} \sum_{k\in K} c_{ij} x_{ij}^k$$

$$\text{subject to} \quad \sum_{j:(i,j)\in E} \sum_{k\in K} x_{ij}^k = 1 \qquad \forall i \in P \cup D \tag{1}$$

$$\sum_{i\in P} \sum_{j:(i,j)\in E} d_i x_{ij}^k \le Q \qquad \forall k \in K \tag{2}$$

$$\sum_{i\in D} \sum_{j:(i,j)\in E} d_i x_{ij}^k \le Q \qquad \forall k \in K \tag{3}$$

$$\sum_{j:(h,j)\in E} x_{hj}^k = 1 \qquad \forall h \in \{o_1, o_3\}, k \in K \tag{4}$$

$$\sum_{i:(i,h)\in E} x_{ih}^k - \sum_{j:(h,j)\in E} x_{hj}^k = 0 \qquad \forall h \in P \cup D, k \in K \tag{5}$$

$$\sum_{j:(j,h)\in E} x_{jh}^k = 1 \qquad \forall h \in \{o_2, o_4\}, k \in K \tag{6}$$

$$s_j^k \ge s_i^k + c_{ij} - M(1 - x_{ij}^k) \qquad \forall (i,j) \in E, k \in K \tag{7}$$

$$a_i \le s_i^k \le b_i \qquad \forall i \in N, k \in K \tag{8}$$

$$u_i^k - r_i^k = \sum_{j\in P\cup\{o_2\}} x_{ij}^k - \sum_{j\in D\cup\{o_4\}} x_{i+n,j}^k \qquad \forall i \in P, k \in K \tag{9}$$

$$u_i^k + r_i^k \le 1 \qquad \forall i \in P, k \in K \tag{10}$$

$$\frac{1}{M} \sum_{i\in P} u_i^k \le g_k \le \sum_{i\in P} u_i^k \qquad \forall k \in K \tag{11}$$

$$t_k = s^k_{o_2} + Ag_k + B\sum_{i\in P} d_i u^k_i \qquad \forall k \in K \qquad (12)$$
$$w_k \geq t_k \qquad \forall k \in K \qquad (13)$$
$$w_k \geq v_i - M(1 - r^k_i) \qquad \forall i \in P, k \in K \qquad (14)$$
$$v_i \geq t_k - M(1 - u^k_i) \qquad \forall i \in P, k \in K \qquad (15)$$
$$\frac{1}{M}\sum_{i\in P} r^k_i \leq h_k \leq \sum_{i\in P} r^k_i \qquad \forall k \in K \qquad (16)$$
$$s^k_{o_3} = w_k + Ah_k + B\sum_{i\in P} d_i r^k_i \qquad \forall k \in K \qquad (17)$$
$$x^k_{ij}, u^k_i, r^k_i, g_k, h_k \in \{0,1\} \qquad \forall i \in P, (i,j) \in E, k \in K \qquad (18)$$
$$s^k_i, t_k, w_k \geq 0 \qquad \forall i \in N, k \in K \qquad (19)$$
$$v_i \geq 0 \qquad \forall i \in P. \qquad (20)$$

The objective is to minimize the total distance traveled. The constraints consist of two parts: vehicle routing (constraints (1) to (8)) and consolidation decisions at the cross-dock (constraints (9) to (17)).

Both the pickup part and the delivery part can be formulated as VRPTWs. Constraints (1) ensure that each node is visited once by one vehicle. Constraints (2) and (3) ensure that for each vehicle, the load on the pickup route and on the delivery route does not exceed the vehicle capacity. Constraints (4) state that each vehicle's pickup route must depart from o_1 and delivery route must leave from o_3. Constraints (5) are flow conservation constraints. Constraints (6) force each vehicle to return to o_2 on its pickup route and return to o_4 on its delivery route. Constraints (7) compute the traveling time between two nodes if they are visited consecutively by the same vehicle. Constraints (8) ensure that each node is visited within its time window and the whole operation is completed within the time horizon.

For the consolidation decisions at the cross-dock, whether a vehicle k should unload or reload product i depends on its pickup and delivery routes. This dependence, which shows the linkage between the pickup part and the delivery part, is expressed by constraints (9) and (10). In these two constraints, the following three cases are considered: if vehicle k picks up i but does not deliver $i+n$, then it unloads the product at the cross-dock; if vehicle k does not pick up i but delivers $i+n$, then it needs to reload the product at the cross-dock; if the vehicle neither picks up i nor delivers $i+n$, then it neither unloads nor reloads the product. These cases are summarized in Table 1.

Constraints (11) to (17) define the internal working flows and deadlines for all the vehicles at the cross-dock. Constraints (11) force g_k to be 1 if the vehicle needs to unload. Constraints (12) indicate that the unloading duration for a vehicle k consists of a fixed time (A) for the preparation of

k picks up i	k delivers $i+n$	k unloads i	k reloads i
$\sum_{j\in P\cup\{o_2\}} x_{ij}^k$	$\sum_{j\in D\cup\{o_4\}} x_{i+n,j}^k$	u_i^k	r_i^k
0	0	0	0
1	1	0	0
1	0	1	0
0	1	0	1

Table 1: Relationship between x_{ij}^k and u_i^k, r_i^k

unloading, and the time for unloading the products, equal to the unit time for unloading a pallet (B) multiplied by the number of pallets ($\sum_{i\in P} d_i u_i^k$) to be unloaded from the vehicle. Constraints (13) and (14) ensure that a vehicle cannot start reloading until it finishes unloading, and all the products to be reloaded on it are ready. The ready time of product i is represented by constraint (15), which depends on the time at which the pickup vehicle of product i finishes unloading. Constraints (16) and (17) for the reloading are similar to (11) and (12).

This formulation contains $O(n^2)$ binary variables, $O(mn)$ continuous variable and $O(n^2m)$ constraints. Without constraints (9) to (16), the model is essentially the problem of solving two independent VRPTWs, abbreviated as 2-VRPTW. It is obvious that any optimal solution to this 2-VRPTW provides a lower bound for the VRPCD. The difficulty in the VRPCD is that the pickup and delivery routes are not independent but correlated. This correlation results from the fact that the same vehicles need to first pick up and then deliver products within the time windows. Therefore, the search for an optimal solution is not only to find shortest routes for both operations, but also to coordinate the exchanges of products at the cross-dock so that all time windows and the time horizon are respected. These two aspects usually conflict with each other. The impact of this conflict on the VRPCD solution will be illustrated in the computational experiments section.

Heuristics

Tabu search (TS) has proven to be one of the best available heuristic methods for solving VRPs, producing high quality solutions within a reasonable amount of computing time (Cordeau *et al.*, 2002). The basic idea of TS is to locally and repeatedly modify a solution while memorizing the modifications to avoid cycling. Modifications to their attributes are stored in a *tabu list* that forbids their use for a certain number of iterations.

In this paper, we develop a TS algorithm for the VRPCD. Applying TS to a new problem requires taking the specific knowledge of the problem into consideration. In the VRPCD, due to the consolidation at the cross-dock, computing the cost of even a simple insertion is very expensive.

To alleviate the computational burden, properties of insertions are investigated and new accelerating strategies are proposed, which have been proved to be very effective. Two neighbourhoods are used alternately in the TS, which is finally embedded within an adaptive memory procedure (AMP). This enables the algorithm to reach good and robust solutions by repeating the TS from different good starting points.

The AMP is described next, followed by the proposed TS algorithm, and by a description of an efficient implementation of local search.

Adaptive memory procedure

In an AMP, a set of vehicle tours is stored in an adaptive memory (AM). A vehicle tour is defined as a pickup route and a delivery route operated by the same vehicle. An initial TS solution is constructed by combining the selected vehicle tours, where the selection preference is probabilistically biased toward tours with good objective values. This idea was first proposed by Rochat and Taillard (1995) in the context of the VRP and of the VRPTW and has been proved to be very effective in providing high quality solutions for related problems.

In an AMP, an improved solution identified during the TS is considered. The vehicle tours in this solution are labeled with the value of the objective function and are included in the AM. Concurrently, the same number of tours with the highest label are removed from the AM. Consequently, the AM consists of a constant number of vehicle tours throughout the algorithm. Algorithm 1 shows how to generate the initial TS solution from the AM. The probability assigned to a specific vehicle tour is $P(r) = (\max\{l_r | r \in AM)\} - l_r)/\sum_{r\in AM} l_r$, where l_r is the label of the r^{th} vehicle tour. A tour with a smaller value of l_r will be selected with a higher probability. At each iteration one vehicle tour is selected and all incompatible tours are removed since each node can be served by only one vehicle. The selecting procedure stops when there are no more vehicle tours compatible with those selected. Finally, the unvisited nodes are assigned to empty vehicles (see lines 8 to 12).

Tabu search heuristic

Our TS algorithm is based on that of Cordeau, Laporte and Mercier (2001), in which infeasible solutions are allowed during the search. The cost function of a solution s is defined by $f(s) = c(s) + \alpha q(s) + \beta d(s) + \gamma w(s)$, where

Algorithm 1 Generate initial solution from the memory

```
 1: $AM' \leftarrow AM$
 2: assign a probability to each element in $AM'$
 3: while $AM'$ is not empty do
 4:     select a vehicle tour $r$ from $AM'$
 5:     delete the routes that cover a node covered by vehicle tour $r$
 6: end while
 7: let $N_{left}$ be the unserved nodes sorted in increasing order of radial angle
 8: while $N_{left}$ is not empty do
 9:     assign the nodes in $N_{left}$ to an empty vehicle $v'$ one by one until the vehicle capacity is reached
10:     remove from $N_{left}$ the nodes that are assigned to $v'$
11: end while
```

$$c(s) = \sum_{k\in K} \sum_{(i,j)\in E} c_{ij} x_{ij}^k; \tag{21}$$

$$q(s) = \sum_{k\in K} \left[\left(\sum_{i\in P} \sum_{j\in P\cup\{0_2\}} d_i x_{ij}^k - Q \right)^+ + \left(\sum_{i\in D} \sum_{j\in D\cup\{0_4\}} d_i y_{ij}^k - Q \right)^+ \right]; \tag{22}$$

$$d(s) = \sum_{k\in K} \left(s_{o_4}^k - b_0 \right); \tag{23}$$

$$w(s) = \sum_{k\in K} \left[\sum_{i\in P} \left(\sum_{j\in P\cup\{o_2\}} s_i^k x_{ij}^k - b_i \right)^+ + \sum_{i\in D} \left(\sum_{j\in D\cup\{o_4\}} s_i^k x_{ij}^k - b_i \right)^+ \right], \tag{24}$$

where $(x)^+ = \max\{0, x\}$. In these equations, $c(s)$ is the total distance traveled by all vehicles, $q(s)$ is the total excess quantity in both the pickup and the delivery parts, $d(s)$ and $w(s)$ are the excess duration and total time window violations, respectively. Thus, if s is feasible, then $f(s) = c(s)$. The coefficients α, β and γ are positive self-adjusting penalties. At each iteration, the values of α, β and γ are modified by a factor $1 + \delta > 1$: if the current solution is feasible with respect to quantity (resp. duration, time windows), the value of α (resp. β, γ) is divided by $1 + \delta$; otherwise, it is multiplied by $1 + \delta$.

In the TS, an insertion moves a node i from its original vehicle k to another vehicle k', as illustrated in Figure 4.

In some TS implementations (e.g., Barbarosoglu and Ozgur, 1999; Tang and Miller-Hooks, 2005), the solution improvement phase is divided into two stages: exploring a small number of moves (namely, small neighbourhood search), and exploring a large number of moves (namely, large neighbourhood search). Empirical results show that alternating between small and large neigh-

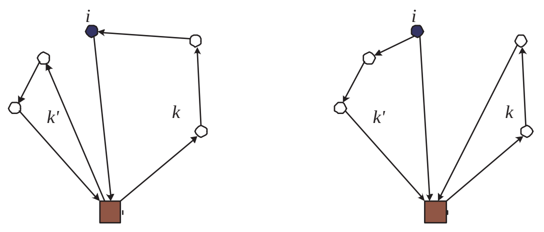

Figure 4: Insertion move

bourhood search enables the search to evolve in an efficient way without degrading solution quality.

In our TS algorithm, the same strategy is exploited. In the large neighbourhood search, every node is moved to every position of every other vehicle. The complexity for this neighbourhood is $O(n^2)$. In the small neighbourhood search, instead of searching the whole solution space, we select a subset of nodes N' ($N' \subseteq N$) and move each of them to a specific subset of vehicles. For instance, a node i' ($i' \in N'$) is tentatively moved to each vehicle in M'_i ($M'_i \subseteq M$). The ways of selecting N' and M'_i are as follows: if in the current solution, a vehicle violates the capacity constraint, exceeds the duration limit or involves any node not visited within its time window, all the nodes served by this vehicle will be added into N' since the infeasibility of the solution will probably be reduced by removing a node from an infeasible route. After collecting all the nodes from all the infeasible vehicle routes, if the size of N' is still less than a fixed number μ, we randomly select $\mu - |N'|$ nodes and add them into N' to make sure that N' is not too small or empty when the solution is nearly feasible or feasible. When considering moving a node $i' \in N'$ to vehicles in M'_i, we would like M'_i to be a set of vehicles that are very close to node i' in order to minimize the total traveled distance. We sort all the nodes $i \in N$ except i' in ascending order of the distance between i and i'. The set M'_i consists of all vehicles that cover any of the ν nodes nearest to i'. Hence, each node in N' is moved to several of its nearest vehicle routes. With fixed values of μ and ν, the running time is independent of the size of the data.

The strategy we used for switching between small and large neighbourhoods is the following: the TS starts with the small neighbourhood and switches to the large neighbourhood if there has been no improvement in the best solution in the last η iterations; when exploring the large neighbourhood, if the best solution is updated within σ iterations, the search process switches back to the small neighbourhood, otherwise the TS stops. The overall structure of TS is presented in Algorithm 2.

As for short and long term memories, we use the same rules as in Cordeau, Laporte and Mercier (2001). To avoid cycling, if a node is removed from a vehicle, reinserting it in that vehicle is forbidden for the next θ iterations unless the move satisfies an *aspiration criterion*, i.e. a feasible

Algorithm 2 Tabu search algorithm

```
1: $Stop$ = FALSE; $Nb$ = 2; $i$ = 0; $s^*$ = $\emptyset$; $f(s^*) = \infty$
2: while $Stop$ $==$ FALSE do
3:    if $Nb == 2$ then
4:       Small neighbourhood search
5:    else
6:       Large neighbourhood search
7:    end if
8:    $s$ = solution found by the neighbourhood search
9:    if $f(s) < f(s^*)$ and $s$ is feasible then
10:      $s^* = s$; $i$ = 0; $Nb$ = 2
11:   else
12:      $i$++
13:   end if
14:   if $Nb$ == 2 and $i > \eta$ then
15:      $Nb$ = 1
16:   end if
17:   if $Nb$ == 1 and $i > \sigma$ then
18:      $Stop$ = TRUE
19:   end if
20: end while
```

solution better than the present best solution is found. To diversify the search, the move cost Δf is increased by a penalty whose value is proportional to the frequency of the move. For more details, see Cordeau, Laporte and Mercier (2001).

Efficient implementation of local search

Compared with the VRPTW, an insertion move for the VRPCD is computationally much more expensive. Due to the consolidation at the cross-dock, the vehicles are no longer independent of each other. For example, an insertion move that shifts a supplier i from vehicle k to k' will not only affect the delivery route of k and k', but will also affect other related vehicles that serve the corresponding customers of the suppliers served by vehicles k and k'.

In order to accelerate the algorithm, we have studied the cost impact of a move. Denote the current solution by S_1, the solution after the move $i : k \rightarrow k'$ by S_2, and the move cost by Δf. The change in total traveled distance is denoted as Δc. The changes in violation of quantity, duration and time windows are denoted as Δq, Δd and Δw, respectively. The following property holds:

Property *If S_1 is feasible, then $\Delta f \geq \Delta c + \alpha \Delta q$.*

Proof. Since $f(s) = c(s) + \alpha q(s) + \beta d(s) + \gamma w(s)$, the cost of a move can be represented as $\Delta f = f(S_2) - f(S_1) = \Delta c + \alpha\Delta q + \beta\Delta d + \gamma\Delta w$. Since S_1 is feasible, then $\Delta d \geq 0$ and $\Delta w \geq 0$ hold. Hence, $\Delta f \geq \Delta c + \alpha\Delta q$. $\square$

Note that to calculate $\beta\Delta d$ and $\gamma\Delta w$, we need to exploit the complete information about S_2 including the duration of each vehicle route and the visiting time of each node; on the other hand, Δc and $\alpha\Delta q$ can be very easily calculated without investigating the details in S_2. Let Δf^* denote the minimum move cost among all moves considered. In the neighbourhood search of a currently feasible solution, a move for which $\Delta c + \alpha\Delta q \geq \Delta f^*$ can be skipped to save computation time. We call this kind of skipping a *conservative skip*.

Preliminary tests have shown that very little computing time is saved by using the *conservative skip* since it only applies to feasible solutions. To reduce the computational time further, we have developed an *aggressive skip* which applies when the route of vehicle k is feasible with respect to the duration and time window constraints. This skip is reasonable because removing i from k is very likely to reduce the violation of route k, and inserting i into route k' may increase the violation of route k'. If vehicle k is already feasible, Δd and Δw are probably positive. The move is therefore skipped if $\Delta c + \alpha\Delta q \geq \Delta f$. Nevertheless, if route k is infeasible with respect to the duration and time window constraints, then the move should not be skipped.

Computational Experiments

Our TS heuristic was implemented in C and executed on a Linux computer with a 2.2GHz Dual Core AMD Opteron$^{\text{tm}}$ Processor 175 and 2 Gbytes of RAM. Due to the practical constraints, for a data set with 200 pairs of nodes, the users would expect to solve the problem relatively quickly. As a result, the computational time of running the algorithm in this paper is limited to five minutes.

Data

The data used in this paper were generated from a real data set belonging to Transvision, a Danish logistics consultancy based in Copenhagen. The real data are confidential and could not be provided to us. The test data consist of five Euclidean sets, denoted by 200a, 200b, 200c, 200d and 200e, respectively, where 200 stands for the number of supplier-customer pairs. Each set consists of suppliers and customers with pickup and delivery locations (x, y) in meters. The time window for each node is two hours. The time horizon for the whole transportation operation is from 6:00 to 22:00. The demand transported from each pickup location to the corresponding delivery location is given in number of pallets. Vehicles drive at a constant speed of 60 km/h and have a capacity of 33 pallets. It takes ten minutes to prepare a vehicle, plus an additional one minute for each pallet to be loaded or unloaded. The locations of the suppliers, the customers and the cross-dock for one data set are given in Figure 5. In this data set, the depot is located in Glostrup, near Copenhagen. The pickup points are mostly in Zealand where Glostrup is situated, and the delivery points are mostly in Jutland.

For preliminary testing purposes, small data sets with 20, 30, 50, 100 or 150 pairs of nodes were generated by randomly selecting the corresponding number of supplier-customer pairs. All the test data can be accessed via the Internet at http://www2.imm.dtu.dk/∼mw/vrpcdData/.

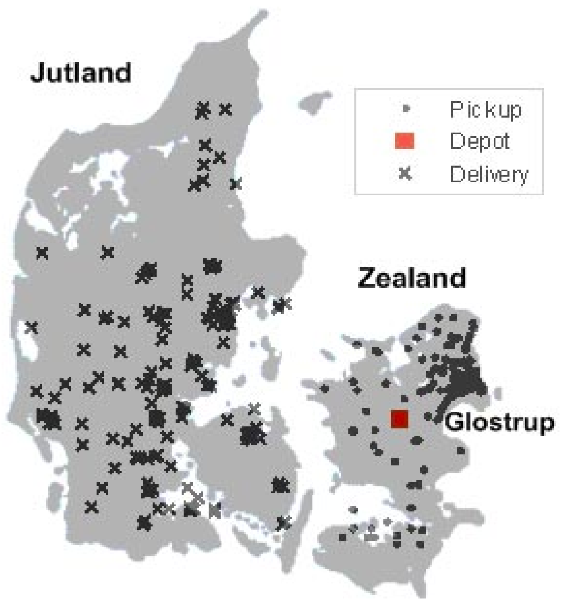

Figure 5: Locations of pickup and delivery nodes for one instance of the VRPCD

Parameter tuning

The algorithm employs a set of parameters whose values require tuning before the algorithm is assessed. In Table 2, these parameters are listed and explained. Based on a large number of runs, the following set of parameters was finally selected: (δ, η, σ, φ, μ, ν) = (3, 1200, 800, 10n, n, 10), where the n is the number of supplier-customer pairs.

In the AMP, it was found that the performance of the algorithm is not very sensitive to the AM size φ. Preliminary results have shown that φ = 10n is large enough. The selection of the number of AMP iterations is a tradeoff between performance and computational cost. Given the running time of five minutes, three iterations are found to be suitable, which is in line with the setting used in Tang and Miller-Hooks (2005).

δ:	the maximum number of AMP iterations;
η:	the maximum number of non-improving iterations in small neighbourhood search in TS;
σ:	the maximum number of non-improving iterations in large neighbourhood search in TS;
φ:	the size of the AM;
μ:	the number of the selected nodes for the small neighbourhood;
ν:	in the small neighbourhood, a selected node i is moved to the vehicles that cover the ν nodes nearest to i;

Table 2: Parameters of the algorithm

As mentioned in the TS description, the alternate use of two neighbourhoods in the TS makes it possible for the search process to move out of the current local optimum. This strategy has already been proved to be very effective in providing high quality solutions as stated in Tang and Miller-Hooks (2005). The same effect is achieved by our algorithm, as shown in Table 3. In the tests, the value of ν is 10 for the small neighbourhood and n for the large neighbourhood. Table 3 illustrates the comparison between two-neighbourhood search and one-neighbourhood search. The first two columns are the data set descriptor and the number of supplier-customer pairs. Columns 3 and 4 report the average solution value over 25 runs and the average computational time in seconds for the two neighbourhood strategies adopted in our algorithm. Columns 5 and 6, and columns 7 and 8 provide the corresponding results for the large neighbourhood search and the small neighbourhood search, respectively. According to Table 3, the two-neighbourhood strategy consistently outperforms the other two one-neighbourhood strategies both in terms of average solution value and computational time.

		Two-neighbourhood		Large neighbourhood		Small neighbourhood	
Data set	n	Average solution value	Average time (seconds)	Average solution value	Average time (seconds)	Average solution value	Average time (seconds)
50a	50	6534.2	16	6568.1	27	6558.7	11
100a	100	12982.9	63	13096.6	93	13265.9	20
100b	100	14770.9	56	14864.9	92	14919.6	20
150a	150	19871.3	139	19939.7	232	20304.5	38
150b	150	21284.0	125	21374.3	218	21537.8	36
200a	200	27684.8	274	27959.9	482	28107.7	73
200b	200	27989.1	278	28241.2	517	28393.3	77

Table 3: Comparison of two-neighbourhood strategy and one-neighbourhood strategy

A large number of trials have shown that the selection of η, σ and μ is not critical over a wide range.

The values 1200, 800 and n are selected based on the results and experience. The parameters α, β, γ are set as in Cordeau, Laporte and Mercier (2001).

Analysis of the main results of the algorithm

We now analyze the impact of the efficient implementation and the correlation among vehicles on the behaviour of the algorithm, and we present results on the VRPCD instances.

Effect of the efficient implementation of the local search

The new *aggressive skip* described in Section 4 is vital to efficiently narrow down neighbourhood size and to reach high quality solutions within short computing times. In theory, the ruling out process may cut off useful moves and degrade the solution quality slightly. However, it works very well in practice.

A comparison between the results with *aggressive skip* and without *aggressive skip* (namely, *full search*) is provided in Table 4. The column $'$Gap$'$ presents the percentage gap between the two results. It is calculated as $100(\bar{z}_{aggressive\_skip} - \bar{z}_{full\_search})/\bar{z}_{full\_search}$.

The column $'$Gap$'$ shows that the TS with *aggressive skip* performs as well as *full search*. The Gap is less than 0.3% for all the tests in Table 4. However, it can be seen, the computational time for the *aggressive skip* is much shorter than for the *full search*. Figure 6 shows the value of the best solution found as a function of the running time of the algorithm for data set 200a. As we can see, the *aggressive skip* significantly speeds up the algorithm while maintaining almost the same solution quality.

		Aggressive skip		Full search		
Data Set	n	Average solution value	Average time (seconds)	Average solution value	Average time (seconds)	Gap (%)
100a	100	12981.9	63	12980.3	891	0.012
100b	100	14770.9	56	14769.3	805	0.011
100c	100	14145.1	57	14125.3	749	0.140
150a	150	19871.3	139	19885.5	2790	-0.071
150b	150	21284.0	125	21265.1	2565	0.089
150c	150	20320.5	140	20326.7	2578	-0.031
200a	200	27683.9	273	27676.2	6519	0.028
200b	200	27989.1	278	27916.5	6449	0.260
200c	200	26654.1	282	26640.3	6053	0.052

Table 4: Comparison of TS with and without *aggressive skip*

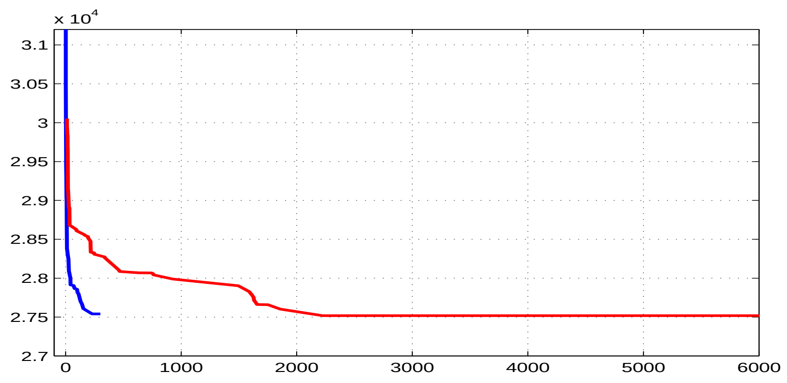

Figure 6: The effect of *aggressive skip*. The short and long curves show the results with and without *aggressive skip*, respectively

Impact of the correlation among vehicles on the VRPCD solution

To illustrate the impact of correlation among vehicles on the VRPCD solution, we vary the preparation time A and the time B for reloading and unloading a pallet. As A and B increase, it becomes more time consuming to exchange products among vehicles at the cross-dock. Restricted by the time windows at the delivery nodes, the algorithm will try to avoid unloading and reloading products and thus hinder the optimization of distances traveled on both sides.

We have applied our algorithm with different settings of A and B. The results are presented in Table 5. Five data sets (20a, 20b, 30a, 100a and 200a) were tested with different settings of A and B. The setting A-B denotes the number of minutes required for A and B, respectively. For each data set and A-B setting, the average objective value over 20 random runs and the average number of nodes whose products are unloaded at the cross-dock are given. ′INF′ means no feasible solution was found in the 20 runs.

As expected, the best objective value the algorithm can find increases with A and B. For large A and B, it is obvious that the optimal 2-VRPTW solutions do not yield good lower bounds for the VRPCD.

It should be stressed that other factors could affect the linkage of the pickup and delivery parts, such as the geographical distribution of suppliers and customers, the time windows, and the number of supplier-customer pairs. When the distribution of suppliers is very different from that of customers, for example when nearby suppliers have far away corresponding customers, when the time windows are short, or when the number of suppliers and customers is large, the correlation between the two parts tends to be strong, i.e., the optimal solution of the VRPCD tends to be farther away from that of the 2-VRPTW.

			A-B				
Data set	n		10-1	15-3	20-5	20-10	30-20
20a	20	Average solution value	2671.6	2674.9	2684.9	2748.8	3010.6
		Nb. unloads	17	13	11	5	2
20b	20	Average solution value	3236.8	3244.6	3285.1	3723.4	4008.2
		Nb. unloads	17	14	8	4	2
30a	30	Average solution value	3923.4	3936.4	4005.5	4156.9	INF
		Nb. unloads	26	21	15	6	INF
100a	100	Average solution value	12981.9	13318.3	13656.2	14640.9	20479.4
		Nb. unloads	96	89	61	37	21
200a	200	Average solution value	27683.9	28425.9	29427.5	33451.9	INF
		Nb. unloads	196	181	124	80	INF

Table 5: The effect of parameters A and B

Results on VRPCD instances

Tables 6 and 7 show the results obtained for small and large instances, respectively. Each instance was run 25 times randomly. In both tables, the average solution value, average computational time and the best solution value over 25 runs are provided. The algorithm of Kallehauge, Larsen and Madsen (2006) was used to solve the corresponding 2-VRPTWs as the lower bounds to the VRPCD.

Data set	n	Average solution value	Average time (seconds)	Best solution value	LB1	Gap (%)
20a	20	2671.6	5	2668.8	2668.793	0.11
20b	20	3236.8	4	3230.9	3228.816	0.25
20c	20	2643.1	4	2632.0	2631.791	0.43
20d	20	3647.8	3	3646.5	3638.727	0.25
20e	20	2834.7	3	2819.1	2816.152	0.66
20f	20	3487.5	5	3483.8	3483.459	0.12
20g	20	3329.3	4	3323.6	3322.730	0.20
20h	20	3428.8	3	3428.5	3428.457	0.01
20i	20	2861.8	4	2861.4	2861.447	0.01
20j	20	2851.7	4	2851.7	2841.704	0.35
20k	20	2903.8	4	2903.8	2883.691	0.70

Table 6: Experimental results for small instances

For every small instance, the 2-VRPTW is solved to optimality and the solution is given in column

		Results			No limit test results			
		Average solution	Average time	Best solution	Average time	Best known solution		Gap2
Data set	n	value	(seconds)	value	(seconds)	value	LB2	(%)
30a	30	3923.4	9	3908.2	1787	3884.7	3757.04	4.02
30b	30	4901.0	7	4855.6	1319	4824.1	4795.65	1.25
30c	30	5146.8	7	5125.2	1495	5112.4	4968.30	3.16
30d	30	3891.9	8	3865.0	1729	3850.0	3708.37	4.22
30e	30	5084.4	7	5041.4	1468	5014.3	4913.24	2.61
50a	50	6534.2	17	6497.3	3865	6471.9	6340.90	2.47
50b	50	7504.9	19	7466.3	3185	7410.6	7201.89	3.67
50c	50	7440.0	20	7350.5	3269	7330.6	7241.05	1.51
50d	50	7107.6	20	7074.0	3658	7050.3	6887.93	2.70
50e	50	7629.4	16	7571.5	3159	7516.8	7347.54	3.05
100a	100	12981.9	63	12878.0	11543	12860.8	12555.57	2.57
100b	100	14770.9	56	14646.8	9967	14526.1	14200.48	3.14
100c	100	14145.0	57	14056.4	10677	13967.8	13631.24	3.12
100d	100	13949.6	57	13844.4	11177	13763.3	13395.33	3.35
100e	100	14396.1	63	14300.4	10643	14212.7	13745.60	4.04
150a	150	19871.3	139	19784.0	24326	19537.3	19012.02	4.06
150b	150	21284.0	125	21098.1	24461	20974.8	20371.08	3.57
150c	150	20320.5	139	20166.2	23754	20126.5	19419.55	3.84
150d	150	20891.3	123	20747.2	24468	20549.4	20013.37	3.67
150e	150	20034.6	140	19888.5	23400	19848.5	19141.66	3.90
200a	200	27683.9	273	27537.4	46586	27324.4	26538.53	3.76
200b	200	27989.1	278	27851.7	43653	27637.7	26722.88	4.22
200c	200	26654.1	282	26472.5	46389	26358.6	25607.31	3.38
200d	200	28088.2	296	27935.3	46615	27749.7	26969.42	3.58
200e	200	26868.6	275	26703.4	45649	26620.6	25776.01	3.60

Table 7: Experimental results for large instances

′LB1′ in Table 6. The gap between the ′Average solution value′ and ′LB1′ is given in column ′Gap′ in Table 6. We can conclude that the algorithm can produce near optimal solutions (less than 1% away from the optimum) within very short computing times (less than 5 seconds) for all the small instances.

For the large instances, as 2-VRPTW itself is an NP-hard problem, it is very difficult to obtain the optimal solution. Instead, we use a lower bound equal to the LP relaxation value computed at the root node of the 2-VRPTW, in column ′LB2′ in Table 7. The gap between the ′Average solution value′ and ′LB2′ is given in column ′Gap2′ in Table 6. As can be seen from the table, Gap2 is consistently below 5% for any data size in the tests. We consider this percentage to be very satisfactory since it overestimates the true optimality gap. We also provide the best known solution values and the corresponding computational time in columns ′No limit test results′ in Table 7. These results are obtained by removing the limit on the computational time and setting the parameters as (δ, η, σ, φ, μ, ν) = (50, 15000, 15000, 10n, 2n, 10).

Conclusion

We have considered the *Vehicle Routing Problem with Cross-Docking* in which the goods from the suppliers and customers must be consolidated at a cross-dock terminal before being dispatched to the customers. The problem was modeled and then solved by means of an efficient heuristic embedding tabu search within an adaptive memory procedure.

Since the cross-dock allows the transfer of goods between vehicles, the pickup and delivery vehicles are not independent of each other. The pickup and delivery parts are also correlated. As a result of these interactions, calculating and performing a move can be difficult. A new *aggressive skip* procedure introduced in the tabu search plays a key role in effectively narrowing down the number of moves to be calculated thoroughly and in reaching high quality solutions within short computing times. The proposed algorithm was tested on realistic data sets involving up to 200 pairs of nodes. Computational results show that it can provide high quality solutions (less than 5% away from the optimum) within very short running times.

Acknowledgements

This work was partially supported by the Canadian Natural Sciences and Engineering Research Council under grants 227837-04 and 39682-05. This support is gratefully acknowledged. Thanks are also due to Brian Kallehauge for computing the 2-VRPTW solutions, and to Jakob Birkedal Nielsen from Transvision for providing the VRPCD data.

References

[1] Apte, U.M., Viswanathan, S., 2000. Effective Cross-Docking for Distribution Efficiencies, *International Journal of Logistics: Research and Applications*, 3, 291-302.

[2] Barbarosoglu, G., Ozgur, D., 1999. A tabu search algorithm for the vehicle routing problem, *Computers & Operations Research*, 26, 255-270.

[3] Bartholdi III, J.J., Gue, K.R., 2004. The best shape for a cross-dock, *Transportation Science*, 38, 235-244.

[4] Chen, P., Guo, Y., Lim, A., Rodrigues, B., 2006. Multiple crossdocks with inventory and time windows, *Computers & Operations Research*, 33, 43-63.

[5] Cook, R.L., Gibson B., MacCurdy, D., 2005. A lean approach to cross-docking, *Supply Chain Management*, 9, 54-59.

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

[7] Cordeau, J.-F., Laporte, G., Mercier, A., 2001. A unified tabu Search heuristic for vehicle routing problems with time windows, *Journal of the Operational Research Society*, 52, 928-936.

[8] Gentry, C.R., 2005. Million-Dollar, *Chain Store Age*, February, 54-56.

[9] Gumus, M., Bookbinder, J.H., 2004. Cross-docking and its implications in location-distribution system, *Journal of Business Logistics*, 25, 199-229.

[10] Jayaraman, V., Ross, A., 2003. A simulated annealing methodology to distribution network design and management, *European Journal of Operational Research*, 144, 629-645.

[11] Kallehauge, B., Larsen, J., Madsen, O.B.G., 2006. Lagrangian duality applied to the vehicle routing problem with time windows, *Computers & Operations Research*, 33, 1464-1487.

[12] Lee, Y.H., Jung, J.W., Lee, K.M., 2006. Vehicle routing scheduling for cross-docking in the supply chain, *Computers & Industrial Engineering*, 51, 247-256.

[13] Ratliff, H.D., Vate, J.V., Zhang, M., 1999. Network design for load-driven dross-docking systems, Technical Report, The Logistics Institute, Georgia Institute of Technology, Atlanta.

[14] Rochat, Y., Taillard, É.D., 1995. Probabilistic diversification and intensification in local search for vehicle routing, *Journal of Heuristics*, 1, 147-167.

[15] Stalk, G., Evans P., Shulman L.E., 1992. Competing on capabilities: the new rules of corporate strategy, *Harvard Business Review*, 70, 57-69.

[16] Sung, C.S., Song, S.H., 2003. Integrated service network design for a cross-docking supply chain network, *Journal of the Operational Research Society*, 54, 1283-1295.

[17] Tang, H., Miller-Hooks, E., 2005. A TABU search heuristic for the team orienteering problem, *Computers & Operations Research*, 32, 1379-1407.