Approximation algorithms for the Vehicle Routing Problem with pick-up and delivery

Giovanni Righini
Dipartimento di Scienze dell'Informazione
Polo Didattico e di Ricerca di Crema
Università degli Studi di Milano
giovanni.righini@unimi.it

6 July 2000

Abstract

Reverse logistics is concerned with the integration of the forward flow of goods that are produced and distributed with the backward flow of waste and used products that are collected and recycled. In the framework of reverse logistics many classical optimization problems such as the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP) must be revisited and reformulated taking into account simultaneously both goods delivery and waste collection. For the VRP with pick-up and delivery (VRPPD) only constructive greedy approximation algorithms have been proposed so far and they have been designed and analized only for the case in which customers demands are divisible. In this paper I examine some variations of the same algorithms and I present two new constructive algorithms, along with their worst-case analysis and their experimental evaluation. I also present some local search algorithms with simple, complex and variable neighborhood, and I report the outcome of computational tests on their performance and robustness. I indicate possible extensions and improvements such as tabu search, bi-level search and demand splitting neighborhoods. At last I discuss how and to what extent the algorithms presented here can be adapted to the problem with indivisible demands.

Keywords: *reverse logistics, vehicle routing, pick-up and delivery, approximation, local search*

1 Introduction

In the last years environmental concerns have received growing attention: more and more countries have been orienting their legislations and policies to the environmental sustainability of economic processes and considerable investments have been devoted to waste reduction and material recycling. The existence of a backward flow of objects to be collected, stored, disassembled and recycled and the necessity of integrating it with the forward flow of the classical supply chain from production to distribution is a new challenge for O.R. researchers and practitioners. To make the reuse of products and materials more profitable than their disposal, reverse logistics processes must be properly optimized. For an in-depth discussion of reverse logistics from an O.R. viewpoint I refer the reader to the review by Fleischmann et alii [4]. Reverse logistics encompasses all aspects of classical logistics, from production scheduling to inventory management to distribution. In particular the aim of reverse distribution logistics is to optimize the delivery of goods from warehouses to customers and the simultaneous collection of waste or used products from customers to warehouses or specialized recycling sites. A classical problem in distribution logistics is the optimization of the tours of a set of vehicles of given capacity that must deliver goods to a set of customers on a transportation network, starting from and returning to a common depot: this problem is known as the Vehicle Routing Problem (VRP). Starting from this basic version of the VRP many variations have been considered in the literature: oriented networks, multiple depots, time windows imposed by each customer, precedence constraints, and many others. The operations related to waste collection can be incorporated in the VRP in different ways: in the VRP with backhauls ([18] [19]) each vehicle can both deliver goods and collect waste but it must complete all deliveries before starting to collect.

A different approach was followed by Mosheiov [13] who formulated the VRP with pick-up and delivery (VRPPD) without the precedence constraint between the last delivery and the first pick-up. He presented three greedy constructive algorithms based on tour partitioning for the VRPPD with divisible demands, in which each customer can be served by more than one vehicle. In this paper I consider the VRPPD with divisible demands, I suggest some improvements to those algorithms and I present two new algorithms, along with the worst-case analysis and the experimental evaluation of all of them in section 3. In section 4 I present neighborhood definitions that allow the design of local search algorithms, I report the outcome of computational tests on their performances and robustness and I outline possible extensions and improvements. In section 5 I discuss the extension of this study to the problem with indivisible demands.

2 Problem formulation

The VRP with pick-up and delivery (VRPPD) is defined as follows.

Data: a fleet of V vehicles with identical capacity Q, a set of N customers with integer weights representing demands (a positive weight indicates a demand for a pick-up, a negative one a demand for a delivery), the distances between the locations where the customers and the depot are placed. The problem can be formulated on a graph (oriented or not) as well as in the Euclidean plane.
Variables: a solution is a set of V tours visiting the customers, starting from and returning to the depot. When a vehicle visits a customer it is supposed to deliver or to pick-up a variable amount of load.
Constraints: each customer must be completely served, that is the vehicles visiting him must pick-up/deliver an overall quantity equal to the customer demand. Each vehicle is supposed to start from the depot carrying an amount of goods equal to the total amount it must deliver and to return to the depot carrying an amount of waste equal to the total amount it picked-up. In each point along its tour each vehicle cannot carry a total load greater than its capacity.
Objective: the goal is to minimize of the total length of the tours.

An important assumption made in [13] is that demands (of both kinds) can be splitted up, that is a customer can be visited more than once by different vehicles. This is in general not allowed in the solution of the classical VRP, because in practice this is not always possible or profitable, expecially when pick-up and delivery operations require tools (cranes, for instance) with significant and expensive set-up times. On the contrary demand can be splitted up, for instance, when the vehicles are buses transporting people, such as in the children transportation problem described in [12]. Under the divisible demands assumption, the problem can be reformulated considering only customers with demands equal to either 1 or -1. In the remainder such unit demand customers are referred to as *1-customers.*

Mixed-integer programming model. The following mixed-integer programming model for the VRPPD with divisible demands was proposed in [13]. P and D are the sets of pick-up and delivery 1-customers respectively, and M_p and M_d are their cardinalities; 1-customers are numbered from 1 to M, with $M = M_p + M_d$; the depot has number 0. Each distance between 1-customers i and j is indicated by c_{ij}. The capacity of each vehicle is Q and the number of vehicles is V. The mixed-integer programming model is the following.

$$min \quad \sum_{i=0}^{M}\sum_{j=0}^{M} c_{ij}\; x_{ij}$$

subject to:

$$\sum_{i=0}^{M} x_{ij} = 1 \quad \forall j = 1, \dots, M$$

$$\sum_{j=0}^{M} x_{ij} = 1 \quad \forall i = 1, \dots, M$$

$$\sum_{j=0}^{M} x_{0j} = V$$

$$\sum_{k=0}^{M} y_{jk} - \sum_{i=0}^{M} y_{ij} = \begin{cases} 1 & \text{if } j \in P \\ 0 & \text{if } j \in D \\ -M_p & \text{if } j = 0 \end{cases} \quad \forall j = 0, \dots, M$$

$$\sum_{i=0}^{M} z_{ij} - \sum_{k=0}^{M} z_{jk} = \begin{cases} 1 & \text{if } j \in D \\ 0 & \text{if } j \in P \\ -M_d & \text{if } j = 0 \end{cases} \quad \forall j = 0, \dots, M$$

$$y_{ij} + z_{ij} \leq Q\ x_{ij} \quad \forall i, j = 0, \dots, M$$

$$x_{ij} \in \{0, 1\}, \quad y_{ij}, z_{ij} \geq 0 \quad \forall i, j = 0, \dots, M$$

The meaning of the variables is the following: $x_{ij} = 1$ if and only if a vehicle travels from 1-customer i to 1-customer j; y_{ij} is the total amount of picked-up load (waste) that is carried from i to j; z_{ij} is the total amount of load to be delivered (goods) that is carried from i to j. Under the divisible demand assumption the number of variables and constraints is so huge that an exact solution cannot be computed in a reasonable amount of time even for problems with only 50 customers and integer demands in the range $[-10, 10]$.

Strong and weak feasibility. Because of the capacity constraints, customers with negative demand (delivery customers) must be visited as soon as possible, to free space needed to serve customers with positive demand (pick-up customers). In the remainder I use the following definitions: a sequence of customers is *strongly feasible* if and only if it can be visited by a vehicle without violations of the capacity constraint; a sequence of customers is *weakly feasible* if it can be transformed into a strongly feasible one by rearranging the same visits in a different order. These definitions are useful for at least two reasons: first, they allow a classification of tour partitioning algorithms, according to the kind of tours they compute and this also suggests how to design new tour partitioning algorithms; second, they allow the design of local search algorithms, since accepting weakly feasible solutions is a useful way to explore larger neighborhoods, while keeping the search "near" the feasible region.

3 Constructive algorithms

Mosheiov [13] presented three algorithms for the VRPPD: Iterated Tour Partitioning (ITP), due to Haimovic and Rinnooy Kan [7] and to Altinkemer and Gavish [1], Exhausted Iterated Tour Partitioning (EITP) and Full Capacity Iterated Tour Partitioning (FCITP). All of them take in input a Hamiltonian tour (not including the depot) computed disregarding customers demands and partition it in a set of tours, restarting N times from each different customer location and retaining the best among the N solutions obtained. Because of divisible demands, one should have to consider all M 1-customers as possible initial points, but this would imply a significant increase in computing time, since M is usually much greater than N. In algorithm ITP each tour is terminated as soon as the total amount of delivered goods or the total amount of picked-up waste equals the capacity Q. In EITP each tour is terminated when the next 1-customer along the Hamiltonian tour would provoke a capacity constraint violation. Algorithm FCITP skips 1-customers that would provoke violations of the capacity constraint and a tour is terminated only when both delivered and picked-up loads equal the capacity (excepted the last tours that are terminated when no 1-customers are left). In FCITP only weak feasibility is required, that is the order in which 1-customers appear along the Hamiltonian tour can be changed later in each tour. Mosheiov observed that algorithms EITP and FCITP clearly dominate algorithm ITP, but he found no domination between EITP and FCITP, the former being preferable for smaller instances and demand ranges.
Tour partitioning algorithms can be classified according to two criteria they use in the definition of tours: first, they can skip 1-customers or not; second, they can define strongly or weakly feasible tours. EITP does not skip 1-customers and it defines strongly feasible tours; FCITP skips 1-customers and it defines weakly feasible tours. This observation suggests the design of two other algorithms, with complementary characteristics; it also suggests to rename the algorithms. Therefore in the remainder I use the following four names: STRONGCONSEC (similar to EITP), STRONGSKIP, WEAKCONSEC and WEAKSKIP (similar to FCITP).
Since WEAKCONSEC and WEAKSKIP produce weakly feasible solutions, an additional subroutine is required to compute a strongly feasible solution: each weakly feasible tour is followed from its beginning to its end and each 1-customer that does

not provoke infeasibility is served, otherwise he is skipped. Then the tour is followed backward, visiting all skipped 1-customers in reverse order. This method guarantees to find a strongly feasible tour: in particular visiting all delivery customers forward and all pick-up customers backward always achieves strong feasibility. This procedure is polynomial and it computes an approximate solution. The subproblem of rearranging a weakly feasible tour to compute a minimum cost strongly feasible one visiting the same subset of customers is an NP-hard problem in itself, since it is a generalization of the TSP. It is known as the Traveling Salesman Problem with pick-up and delivery (TSPPD) and it has been solved only in a heuristic way by constructive algorithms [12] [3] [2] and more recently by tabu search [6].
A final observation comes from the lack of symmetry between delivery 1-customers (to be served as soon as possible) and pick-up 1-customers (to be served as late as possible). A consequence of this asymmetry is that if the initial Hamiltonian tour is reversed, the same tour partitioning algorithms can produce different (and possibly better) solutions.

3.1 Worst-case analysis

In [13] the author proved worst-case error bounds for EITP and FCITP, that apply to STRONGCONSEC (EITP) and to WEAKSKIP (FCITP) respectively. For each algorithm he obtained an upper bound to the value of the solution combining three upper bounds: an upper bound on the number of vehicles, an upper bound on the total distance covered by each vehicle to travel between the depot and the customers (radial distance term), and an upper bound on the total distance covered by each vehicle to travel from customer to customer (local distance term). A bound on the approximation error is then obtained dividing the upper bound by a lower bound such as $\max\{\frac{1}{Q}\bar{R}M, L(H)\}$, where $\bar{R}$ indicates the average distance between the depot and the 1-customers and $L(H)$ indicates the length of the initial Hamiltonian tour.

STRONGCONSEC (EITP). In STRONGCONSEC the minimum number of 1-customers served by each vehicle is Q, since each tour is strongly feasible; therefore the maximum number of vehicles used is $\lceil\frac{M}{Q}\rceil$. The total radial distance traveled by each vehicle is limited by $2R_{max}$, where R_{max} indicates the maximum distance between the depot and a customer. The total local distance is bounded by the length of the Hamiltonian tour, $L(H)$, since the strongly feasible tours are made of consecutive customers. Therefore the value z^{SC} of a solution computed by STRONGCONSEC is bounded from above by $\lceil\frac{M}{Q}\rceil 2R_{max} + L(H)$.

WEAKSKIP (FCITP). The number of vehicles used is $\max\{\lceil\frac{M_d}{Q}\rceil, \lceil\frac{M_p}{Q}\rceil\}$, since each vehicle (but the last ones) serves Q pick-up 1-customers and Q delivery 1-customers. The local distance traveled by each vehicle is bounded by $2L(H)$, since each weakly feasible tour must never be traversed more than twice (forth and back) to produce a strongly feasible ordering of the customers. The radial distance term is $\max\{\lceil\frac{M_d}{Q}\rceil, \lceil\frac{M_p}{Q}\rceil\}2R_{max}$. Therefore an upper bound to z^{WS} is $\max\{\lceil\frac{M_d}{Q}\rceil, \lceil\frac{M_p}{Q}\rceil\}(2R_{max} + 2L(H))$.

STRONGSKIP and WEAKCONSEC. Following the same method it is possible to prove worst-case error bounds for the new algorithms STRONGSKIP and WEAKCONSEC. In both cases the number of vehicles is bounded by $\lceil\frac{M}{Q}\rceil$ as in STRONGCONSEC. The total local distance in WEAKCONSEC is bounded by $2L(H)$, since weakly feasible tours do not overlap and each of them must not be traversed more than twice; in STRONGSKIP the total local distance is bounded by the maximum number of vehicles times $L(H)$, since each strongly feasible tour is bounded by the length of the Hamiltonian tour and it must be traversed only once. The radial distance term is the same as above, that is the maximum number of vehicles times $2R_{max}$. Therefore an upper bound to z^{WC} is $\lceil\frac{M}{Q}\rceil 2R_{max} + 2L(H)$, whereas an upper bound to z^{SS} is $\lceil\frac{M}{Q}\rceil(L(H) + 2R_{max})$.

These worst-case bounds, however, are not so interesting for practical purposes, since they depend on the data of the instances and they can be very large and very loose. A clearer insight into the behaviour of the four algorithms can be obtained by an experimental evaluation of their relative performances.

3.2 Experimental tests

The aim of the experimental tests was to evaluate: (a) the robustness of the four algorithms to the change of orientation of the Hamiltonian tour; (b) the robustness of the four algorithms to the quality of the Hamiltonian tour; (c) the relative quality of the solutions computed by the four algorithms.
For the tests I used random Euclidean instances generated with the same technique described in [13]: 6 sets of 50 instances, with a number of customers equal to 50 (sets 1, 2, 3) and 100 (sets 4, 5, 6) uniformly distributed in a square centered around the depot, and demands uniformly distributed in ranges $[-10, 10]$, $[-20, 20]$ and $[-40, 40]$ (excluding the null value). This corresponds to an average number of 250, 500, 1000, 500, 1000 and 2000 1-customers.

Orientation of the initial tour. In order to evaluate how much the four algorithms are robust to the orientation of the initial tour, I tested them on all 300 problem instances, using four initial Hamiltonian tours computed by four different approximation algorithms for the TSP: Nearest Insertion [16], Cheapest Insertion [16] (both initialized with the two customers locations closest to each other), Farthest Insertion [16] and Largest Insertion [15] (both initialized with the two customers locations farthest from each other). In [13] the author suggests that the number of restarts of each tour partitioning algorithm can be limited to the number of customers in the first tour obtained starting from an arbitrary customer. Nevertheless, in all my experiments the tour partitioning algorithms were executed with N different starting customer locations, to get rid of tail effects due to the last incomplete tour. Results are reported in table 1. They indicate the maximum and the average difference between the best solutions found reversing the orientation of the Hamiltonian tours. Errors are indicated as percentage of the best values found. Average values are computed on 50 instances in each problem set, and 4 initial tours for each instance.

Table 1: Robustness to Hamiltonian tour reversal

Problem	STRONGCONSEC		WEAKCONSEC		STRONGSKIP		WEAKSKIP	
Set	max (%)	av. (%)	max (%)	av. (%)	max (%)	av. (%)	max (%)	av. (%)
1	12.00	3.40	7.43	2.00	13.81	3.54	10.17	2.76
2	9.93	2.23	3.00	0.80	12.32	3.28	8.51	2.05
3	4.26	1.08	1.60	0.36	15.57	3.75	10.70	1.60
4	8.61	2.61	3.77	1.01	10.23	2.34	7.19	1.74
5	5.20	1.60	1.77	0.47	11.12	2.56	5.96	1.44
6	2.79	0.87	0.93	0.21	13.57	2.47	6.05	1.37

As expected, algorithms requiring strong feasibility are less robust, because strong feasible tours are likely to be quite different if the Hamiltonian tour is reversed. Algorithms that skip 1-customers along the Hamiltonian tour are also very sensitive to its reversal since the length of their tours is heavily influenced by the order of pick-up and delivery 1-customers more than their positions: 1-customers far away from one another can be inserted into the same tour. Not surprisingly algorithm WEAKCONSEC is the most robust, because weakly feasible tours made of consecutive 1-customers are likely to be independent of the orientation. All results reported in the remainder of the paper have been obtained running the tour partitioning algorithms twice with the Hamiltonian tour oriented in both ways and taking the best of the two solutions found.

Quality of the initial tour. To evaluate the relationship between the quality of the Hamiltonian tour and that of the VRPPD solution, I compared, for each problem set, the average length of the Hamiltonian tour and the average length of the corresponding VRPPD solution, for each TSP approximation algorithm and for each tour partitioning algorithm. Results are summarized in table 2, where the names of the algorithms have been shortened, for brevity.

Table 2: Solutions obtained with different initial tours						
Set	TSP alg.	TSP value	SC	WC	SS	WS
1	FI	6.1250	39.1054	36.1520	33.9063	33.4282
1	LI	6.1905	**38.8893**	**36.0474**	34.1436	33.4391
1	CI	6.7719	39.4035	36.8641	34.9236	34.4745
1	NI	6.9307	39.0823	36.4596	34.8416	34.5593
2	FI	6.0610	76.1001	70.1398	58.6724	57.3497
2	LI	6.1513	76.2258	70.7125	58.8302	**57.3110**
2	CI	6.6539	76.3681	70.7478	60.0684	58.6646
2	NI	6.8438	76.3825	71.0200	60.2474	59.0687
3	FI	6.0870	153.3867	145.8742	109.9501	105.1483
3	LI	6.1723	**152.4870**	**144.8925**	**109.8336**	**105.0915**
3	CI	6.6564	153.4110	146.0542	111.4055	106.8562
3	NI	6.8620	153.8527	146.2680	112.0868	107.5753
4	FI	8.4495	73.3342	67.4608	60.7277	59.6869
4	LI	8.6346	73.7086	**67.4428**	60.8500	59.9013
4	CI	9.2652	74.5177	68.4676	62.0421	61.0904
4	NI	9.4902	74.9834	68.8926	62.5290	61.4292
5	FI	8.4671	149.0063	136.1926	111.1349	106.9070
5	LI	8.5833	**148.7661**	136.4488	**111.0553**	107.0815
5	CI	9.2799	149.3819	137.4976	112.9882	109.0575
5	NI	9.6024	149.2983	137.6410	113.6987	109.7195
6	FI	8.4334	302.0261	285.8687	209.1777	199.8138
6	LI	8.5473	**301.9014**	**285.0562**	209.2434	200.0801
6	CI	9.2679	302.1005	286.2663	212.3116	203.4471
6	NI	9.5368	302.4245	285.9151	212.7316	204.1168

The quality of the solutions computed by the tour partitioning algorithms corresponds to the quality of the starting solutions: in particular Farthest Insertion and Largest Insertion are by far the best approximation algorithms to compute a Hamiltonian tour, even if they do not have the approximation properties of Cheapest Insertion used in [13] to prove worst-case bounds. It is worth noting that solutions computed starting from tours obtained by Largest Insertion are often better than solutions computed starting from tours obtained by Farthest Insertion, even if in average Farthest Insertion dominates the other three TSP algorithms. This can be observed in different problem sets and with different tour partitioning algorithms (bolded values).

Solutions quality. Looking at table 2 it is easy to compare the performances of the tour partitioning algorithms. For any initialization and for any problem size, they rank in the same way: WEAKSKIP is definitely the best, followed by STRONGSKIP, WEAKCONSEC and STRONGCONSEC. Since WEAKSKIP and STRONGCONSEC are variations of Mosheiov's algorithms FCITP and EITP, for which no clear domination was reported, these experiments allow to conclude that the modifications proposed here strongly improve the performance of WEAKSKIP with respect to FCITP: the comparison between WEAKSKIP and STRONGCONSEC shows that the former is quite superior also on small instances (problem sets 1 and 4) for which EITP performed better than FCITP according to Mosheiov.

Number of vehicles. The minimization of the number of vehicles is usually a very important issue in vehicle routing problems. It is often more important than the minimization of the total tours length, since the cost of one more vehicle plus the salary of one more driver is usually quite greater than the savings achievable from length minimization. Therefore a correct evaluation of vehicle routing algorithms must take into account this aspect too. The WEAKSKIP algorithm is the best from this viewpoint, because it fully exploits the capacity of each vehicle and therefore it minimizes the number of vehicles required. Table 3 shows the average number of vehicles required by each tour partitioning algorithm on each problem set. Each average value is computed on 50 instances and 4 initial tours for each instance, as before.

Table 3: Number of vehicles used

Problem Set	SC	WC	SS	WS
1	22.205	20.040	16.350	16.160
2	46.300	42.510	30.565	30.280
3	95.420	90.325	58.010	57.540
4	44.010	39.345	30.625	30.200
5	91.950	83.645	58.165	57.560
6	191.735	180.920	113.535	112.760

STRONGSKIP finds solutions with a number of vehicles very close to the minimum, that is given by WEAKSKIP. On the contrary STRONGCONSEC and WEAKCONSEC partition the Hamiltonian tour into a greater number of tours: the number of vehicles in the solutions computed by these two algorithms is between 25 and 70 percent above the minimum.

As a result of these tests with constructive tour partitioning algorithms, WEAKSKIP clearly dominates the other three; therefore I used WEAKSKIP to compute initial solutions for the local search algorithms described in the next section.

4 Local search

In the design of a local search algorithm for the VRPPD it is important to consider that delivery customers are likely to be served earlier and pick-up customers later; consequently, if a non-trivial tour is reversed, it is likely to violate the capacity constraint. Therefore it is advisable to use neighborhood definitions which are effective for the asymmetric version of the VRP and do not imply the reversal of parts of the solution. Three neighborhoods called RELOCATE, EXCHANGE and CROSS for local search algorithms for the asymmetric VRP are illustrated in [8] and [20] and they can be modified and adapted to the VRPPD.

During the local search, a feasible solution is represented by a set of V strongly feasible tours; each tour t is represented by an ordered list of C_t *visits* to customers, indicated by $v_{t,k}$ with $k = 1, \dots C_t$; each visit $v_{t,k} = (i, r_i)$ is represented by a customer index i, ranging from 1 to N and an amount r_i of load picked-up from ($r_i > 0$) or delivered to ($r_i < 0$) customer i. In order to check weak feasibility the total amounts of picked-up and delivered load, p_t and d_t are recorded for each vehicle t. In order to check strong feasibility the following additional pieces of information are associated to each visit $v_{t,k}$: the total amount of load picked-up by vehicle t before and after the visit, indicated by $p^<_{t,k}$ and $p^>_{t,k}$ respectively; the total amount of load to deliver carried by the vehicle before and after the visit, indicated by $d^<_{t,k}$ and $d^>_{t,k}$ respectively; the maximum total load of the vehicle before and after the visit, indicated by $L^<_{t,k} = max_{j \leq k}\{p^<_{t,j} + d^<_{t,j}\}$ and $L^>_{t,k} = max_{j \geq k}\{p^>_{t,j} + d^>_{t,j}\}$ respectively. The following relations hold, assuming $v_{t,k} = (i, r_i)$:

$$p^<_{t,k} + \max\{r_i, 0\} = p^>_{t,k} \quad \forall t = 1, \dots, V, \ \forall k = 1, \dots, C_t$$

$$d^<_{t,k} - \max\{-r_i, 0\} = d^>_{t,k} \quad \forall t = 1, \dots, V, \ \forall k = 1, \dots, C_t$$

$$p^>_{t,k} = p^<_{t,k+1} \quad \forall t = 1, \dots, V, \ \forall k = 1, \dots, C_t - 1$$

$$d^>_{t,k} = d^<_{t,k+1} \quad \forall t = 1, \dots, V, \ \forall k = 1, \dots, C_t - 1$$

$$p^<_{t,1} = 0 \quad \forall t = 1, \dots, V$$

$$p^>_{t,C_t} = p_t \quad \forall t = 1, \dots, V$$

$$d^<_{t,1} = d_t \quad \forall t = 1, \dots, V$$

$$d^>_{t,C_t} = 0 \quad \forall t = 1, \dots, V$$

$$L^<_{t,k} = \max\{L^<_{t,k-1}, p^<_{t,k} + d^<_{t,k}\} \quad \forall t = 1, \dots, V, \ \forall k = 2, \dots, C_t$$

$$L^>_{t,k} = \max\{p^>_{t,k} + d^>_{t,k}, L^>_{t,k+1}\} \ \ \forall t = 1, \ldots, V, \ \forall k = 1, \ldots, C_t - 1$$

$$L^<_{t,1} = d_t \ \ \forall t = 1, \ldots, V$$

$$L^>_{t,C_t} = p_t \ \ \forall t = 1, \ldots, V$$

An important assumption is that customers visits can be neither grouped nor splitted up by the local search algorithm. Exceptions to this rule are discussed in subsection 4.2. In the remainder S indicates the number of customers visits in the solution: its value is between the number N of customers and the number M of 1-customers. For each neighborhood I define hereafter the procedures for the generation of new solutions, their feasibility check, their evaluation and the update of the current solution.

Neighborhood RELOCATE. This neighborhood includes all strongly feasible solutions obtained by deleting a customer visit $v_{t_1,k_1} = (i, r_i)$ from its tour t_1 and reinserting it in a tour t_2.
Generation. A new solution is generated for each customer visit and for each feasible insertion position. It is possible that $t_1 = t_2$ that is the customer visit is relocated in the same tour. The number of solutions considered is $O(S^2)$. To save computing time, tour t_2 is scanned sequentially from the beginning if $r_i < 0$ and from the end if $r_i > 0$ and the scan is terminated as soon as the first infeasible insertion position is reached.
Feasibility Check. It takes constant time. The following two conditions must hold when visit (i, r_i) is inserted between v_{t_2,k_2} and v_{t_2,k_2+1}:

$$L^>_{t_2,k_2} + \max\{r_i, 0\} \leq Q$$

$$L^<_{t_2,k_2+1} + \max\{-r_i, 0\} \leq Q$$

Evaluation. It takes constant time: two arcs are replaced by one, when the visit is deleted from its tour and one arc is replaced by two when the visit is inserted in a new position. The difference between the lenght of inserted arcs and deleted arcs gives the cost of the local search step.
Update. Its time complexity is $O(Q)$, since it implies the recomputation of the partial load values along t_1 and t_2.

Neighborhood EXCHANGE. According to the definition of the EXCHANGE neighborhood, a new solution of an asymmetric VRP is obtained from the current one by selecting two customers and exchanging them. In the VRPPD, the exchange of two visits can produce infeasible, weakly feasible or strongly feasible solutions. Therefore I defined two similar neighborhoods, EXCHANGESTRONG and EXCHANGEWEAK: the former includes only strongly feasible solutions produced by an exchange; the latter includes also weakly feasible solutions.

Neighborhood EXCHANGESTRONG .
Generation. All ordered pairs of visits are considered in lexicographical order. The exchange of two visits is allowed even if they belong to the same tour. The number of solutions explored is thus $O(S^2)$.
Feasibility check. When $v_{t_2,k_2} = (j, r_j)$ replaces $v_{t_1,k_1} = (i, r_i)$ and $t_1 \neq t_2$, the following two conditions are tested on t_1:

$$L^{>}_{t_1,k_1} + \max\{r_j, 0\} - \max\{r_i, 0\} \leq Q$$

$$L^{<}_{t_1,k_1} + \max\{-r_j, 0\} - \max\{-r_i, 0\} \leq Q$$

These conditions are necessary and sufficient for t_1 to remain strongly feasible. An analogous test must be done on t_2 and the feasibility check takes constant time. When $t_1 = t_2$ (suppose wlog that $k_1 < k_2$), the feasibility check is made by scanning the tour from position k_1 to position k_2 and verifying the capacity constraint on each arc in between. In this case the test takes $O(Q)$ time.
Evaluation. The value of the new solution depends only on the arcs involved in the exchange, and therefore it can be computed in constant time: the arcs are in general four, with an exception when $t_1 = t_2$ and $k_2 = k_1 + 1$ (consecutive visits along the same tour).
Update. The update of the data-structures, including the load values for each visit in the two tours, takes $O(Q)$ time.

Neighborhood EXCHANGEWEAK. A neighbor solution belongs to EXCHANGE-WEAK even if the exchange of two visits produces one or two weakly feasible tours and in this case a suitable procedure is executed to recover strong feasibility.
Generation. All ordered pairs of visits $v_{t_1,k_1} = (i, r_i)$ and $v_{t_2,k_2} = (j, r_j)$ are considered. The number of solutions in the neighborhood is $O(S^2)$.
Feasibility check. A tour is weakly feasible if and only if both the total delivered load and the total picked-up load do not exceed Q. Therefore the following conditions are checked for tour t_1:

$$p_{t_1} - \max\{r_i, 0\} + \max\{r_j, 0\} \leq Q$$

$$d_{t_1} - \max\{-r_i, 0\} + \max\{-r_j, 0\} \leq Q$$

Analogous conditions are checked on tour t_2. The test takes constant time.
Evaluation. Before the evaluation, weakly feasible solutions are transformed into a strongly feasible ones. The transformation implies a rearrangement of the visits along each weakly feasible tour. Consider tour t_2 in which a visit (j, r_j) is replaced by (i, r_i). Since the deletion of a visit does never affect strong feasibility, tour t_2 is strongly feasible after the deletion of (j, r_j) and before the insertion of (i, r_i). Therefore a strongly feasible tour can be computed by selecting the best among all feasible insertion positions of (i, r_i), without rearranging the order of the other visits. To shorten the computing time, it is useful to start from the beginning of t_2 if $r_i < 0$ and from the end if $r_i > 0$ and to stop the search as soon as strong feasibility is lost. The time complexity is $O(Q)$ when $t_1 \neq t_2$. If $t_1 = t_2$, it is necessary to explore all pairs of insertion positions and the time complexity of the subroutine is $O(Q^2)$.
Update. The update step takes $O(Q)$ time as in the case of EXCHANGESTRONG.

Neighborhood CROSS. This neighborhood includes all strongly feasible solutions obtained by the replacement of two arcs by other two. It involves two different tours t_1 and t_2; an arc is deleted in each of them dividing it into two parts, that are recombined, so that a new tour is made of the first part of t_1 and the second part of t_2 and another new tour is made of the first part of t_2 and the second part of t_1.
Generation. A new solution is examined for each ordered pair of arcs (a_1, a_2) where a_1 links $v_{t_1,k_1} = (i_1, r_{i_1})$ to $v_{t_1,k_1+1} = (j_1, r_{j_1})$ and a_2 links $v_{t_2,k_2} = (i_2, r_{i_2})$ to $v_{t_2,k_2+1} = (j_2, r_{j_2})$ with $t_1 \neq t_2$. This corresponds again to $O(S^2)$ solutions.
Feasibility Check. The feasibility test is made in constant time checking the following conditions:

$$L^{<}_{t_1,k_1+1} - d^{<}_{t_1,k_1+1} + d^{<}_{t_2,k_2+1} \leq Q$$

$$L^{>}_{t_2,k_2} - p^{>}_{t_2,k_2} + p^{>}_{t_1,k_1} \leq Q$$

$$L^{<}_{t_2,k_2+1} - d^{<}_{t_2,k_2+1} + d^{<}_{t_1,k_1+1} \leq Q$$

$$L^{>}_{t_1,k_1} - p^{>}_{t_1,k_1} + p^{>}_{t_2,k_2} \leq Q$$

Evaluation. The value of an exchange is computed in constant time, since it depends on the lengths of the four arcs involved.
Update. The update of the load values requires $O(Q)$ time on both tours involved in the exchange.

Neighborhood 3-ARC-EXCHANGE. This neighborhood includes all strongly feasible solutions obtained by the recombination of three different tours, in the same way as in CROSS.
Generation. The number of new solutions to examine is $O(S^3)$. For each triple of deleted arcs there are two possible ways of recombining them to produce three different new tours.
Feasibility Check, Evaluation, Update. Same as in CROSS.

Complex and variable neighborhood. When different neighborhoods are available in a local search framework, it is possible to combine them. In particular when all neighborhoods are explored at each local search step, they form a complex neighborhood; when they are used alternatively, so that if one does not provide improving solutions then another one is explored, they form a variable neighborhood. The outcome of the tests reported in the next subsection gives experimental evidence that the combination of the neighborhoods defined above originates local search algorithms with complex or variable neighborhood that yield quite good solutions and are very robust to initialization.

4.1 Experimental tests

The experiments were aimed at: (a) comparing solutions quality and computing time depending on the neighborhood used; (b) evaluating the performance of the local search algorithms based on simple, complex and variable neighborhood; (c) evaluating the robustness of the local search algorithms to initialization. Unless otherwise specified, local search algorithms were initialized with the solution computed by Farthest Insertion and WEAKSKIP. All experiments were performed on a 350 MHz PC. Computing time is indicated in seconds:hundredths.

Simple neighborhood. Table 4 reports the outcome (average values) of the experiments on all 50×6 problem instances, solved using only one neighborhood at a time (simple neighborhood local search). Best values are bolded.

Table 4: Local search with simple neighborhood

Pr.	Initial	RELOCATE		XSTRONG		XWEAK		CROSS		3-ARC-X	
Set	value	value	time	value	time	value	time	value	time	value	time
1	33.4282	31.7606	00:10	31.4015	00:22	**30.8146**	01:06	32.6079	**00:09**	32.7713	01:89
2	57.3497	55.4627	**00:17**	54.4182	00:44	**53.9700**	01:38	55.6834	00:36	55.8700	13:87
3	105.1483	103.2485	**00:41**	100.4645	02:16	**100.2151**	03:61	102.2890	01:62	102.3503	126:08
4	59.6869	57.4924	**00:43**	56.2836	01:66	**55.2043**	07:00	57.9746	00:76		
5	106.9070	103.9711	**00:98**	101.8215	03:69	**101.0511**	08:20	103.5142	03:74		
6	199.8138	195.7997	**03:20**	190.2228	20:45	**189.7608**	31:99	192.6812	17:02		

The algorithm that uses RELOCATE is the fastest, but it does not produce the best solutions; EXCHANGESTRONG, CROSS, EXCHANGEWEAK and 3-ARC-EXCHANGE follow in order of increasing computing time. 3-ARC-EXCHANGE was not even tested on 100 customers instances since the quality of the solutions obtained does not justify the time spent. The most effective neighborhood (in average) is EXCHANGEWEAK.

Complex neighborhood. I used EXCHANGEWEAK, RELOCATE and CROSS to define a complex neighborhood. I did not use EXCHANGESTRONG, because it is smaller than EXCHANGEWEAK, and 3-ARC-EXCHANGE, because it is too time consuming. Results are reported in table 5. The local search with complex neighborhood achieves better solutions than all local searches with simple neighborhoods; on the other hand it is obviously more time consuming.

Table 5: Local search with complex neighborhood

Pr.	Initial	Final	Computing	XWEAK		RELOCATE		CROSS	
set	value	value	time	moves (%)	δ (%)	moves (%)	δ (%)	moves (%)	δ (%)
1	33.4282	29.8142	02:12	**64.0**	**67.6**	21.1	19.9	14.9	12.5
2	57.3497	53.1042	02:62	**62.6**	**66.7**	16.4	14.3	20.9	18.9
3	105.1483	99.4792	08:89	**68.4**	**72.2**	14.4	12.6	17.2	15.2
4	59.6869	53.6316	13:63	**67.6**	**69.4**	14.0	12.0	18.4	18.6
5	106.9070	99.1510	20:24	**59.3**	**57.4**	15.6	16.5	25.1	26.0
6	199.8138	188.5752	72:53	**66.1**	**66.3**	12.0	10.2	21.9	23.4

The last six columns in table 5 report the percentage of local search steps in each neighborhood and the corresponding percentage (δ) of the overall improvement obtained during the search. This gives information about the relative effectiveness of the different neighborhoods. About two thirds of the local search steps are made within the EXCHANGEWEAK neighborhood, while CROSS is slightly more effective than RELOCATE. Moreover the improvement obtained by each EXCHANGEWEAK step (that is the ratio δ / moves) is greater than those obtained by the other two kinds of steps.

Variable neighborhood. In variable neighborhood local search one neighborhood is explored first: if an improving solution is found, the current solution is updated and a new iteration starts; otherwise a second neighborhood is explored, an so on. This approach was applied, for instance, to multi-source location problems [14] [17]. Table 6 reports the results of a variable neighborhood search in which the neighborhoods were explored in the order EXCHANGEWEAK, RELOCATE and CROSS. I put EXCHANGEWEAK first, since the results outlined in table 5 suggest it is the most effective; between RELOCATE and CROSS I gave precedence to the fastest one, that is RELOCATE.

Table 6: Local search with variable neighborhood

Pr.	Initial	Final	Computing	XWEAK		RELOCATE		CROSS	
set	value	value	time	moves (%)	δ (%)	moves (%)	δ (%)	moves (%)	δ (%)
1	33.4282	29.8806	02:04	77.9	84.3	15.3	9.9	6.8	5.8
2	57.3497	53.0111	02:05	79.1	86.1	12.5	7.9	8.4	6.0
3	105.1483	99.5536	05:43	86.8	92.9	7.7	4.1	5.5	3.0
4	59.6869	53.8290	12:76	80.7	86.8	10.8	6.1	8.6	7.0
5	106.9070	99.1865	16:01	80.6	85.5	9.5	6.7	9.9	7.8
6	199.8138	188.5886	57:75	87.8	93.9	6.2	2.8	6.0	3.3

Since not all neighborhoods are explored at each local search step, the variable neighborhood algorithm is faster than the complex neighborhood one. The solutions are in average worse but the difference is very small. On problem set 2 the variable neighborhood algorithm performs even better than the complex neighborhood one. Also the variable neighborhood algorithm is superior to all simple neighborhood algorithms as far as solutions quality is concerned.

Robustness to initialization. The local search algorithms with simple, complex and variable neighborhood were tested with random initialization, taking as initial solution the output of WEAKSKIP with a random Hamiltonian tour in input. Table 7 reports the average values of the solutions obtained with random initialization and the relative errors (ϵ, in percentage) with respect to the values obtained in tables 4, 5 and 6.

Table 7: Local search with random initialization

Pr. Set	Initial value	RELOCATE value	RELOCATE ϵ (%)	XSTRONG value	XSTRONG ϵ (%)	XWEAK value	XWEAK ϵ (%)	CROSS value	CROSS ϵ (%)	Complex value	Complex ϵ (%)	Variable value	Variable ϵ (%)
1	49.59	39.27	23.64	34.44	9.68	32.83	6.56	40.08	22.90	**30.68**	**2.89**	30.82	3.15
2	75.61	64.96	17.13	58.30	7.13	57.19	5.97	64.27	15.42	54.86	**3.31**	**54.79**	3.35
3	129.14	115.39	11.76	104.54	4.06	103.59	3.36	111.61	9.11	101.58	2.11	**101.55**	**2.00**
4	98.35	78.66	36.82	62.26	10.62	59.29	7.40	72.98	25.88	**55.57**	3.62	55.72	**3.50**
5	151.90	129.07	24.14	108.16	6.23	105.45	4.35	119.33	15.27	101.95	2.82	**101.87**	**2.71**
6	257.85	228.55	16.72	196.81	3.46	195.26	2.90	209.93	8.95	192.12	1.88	**192.12**	**1.87**

Local search with simple neighborhood is quite sensitive to initialization: the solutions quality deteriorates to the extent of 37% compared to the well-initialized search. The error is smaller for greater numbers of 1-customers. On the contrary algorithms with complex and variable neighborhoods are very robust and the solutions deterioration due to random initialization remains below 4% and becomes smaller for larger instances.

4.2 Extensions

The algorithms illustrated above can be refined and improved. Hereafter I indicate three ways to design more sophisticated local search algorithms for the VRPPD: tabu search, bi-level search and splitting neighborhoods.

Tabu search. In the scientific literature on routing problems many researchers have reported good results achieved by tabu search (see for instance [6] for the TSP with pick-up and delivery and [5] for the VRP). In tabu search local search steps are performed even if the value of the objective function is worsened. Cycling is prevented by forbidding "undo" moves for a certain number of iterations. The tabu search paradigm can be applied to the VRPPD, by accepting worsening moves and by fixing the customer visits involved in a local search step in the RELOCATE or in the EXCHANGE neighborhood and fixing the arcs involved in a local search step in the CROSS neighborhood. Such customer visits and arcs remain fixed for a certain number of iterations during which they cannot be relocated, exchanged or crossed again. I performed some limited testing with the tabu version of the local search algorithm with complex neighborhood on problem sets 1, 2 and 3, setting the length of the tabu list to 5 and stopping the algorithm after 200 iterations without improvement. The performances of the tabu search were comparable with those of the complex neighborhood local search, even if the computing time required in the former case was definitely greater. However it is possible that a more careful tuning of the tabu search parameters lead to better results.

Bi-level local search. One of the most effective local search algorithms for the (A)TSP is that of Lin and Kernighan [9], that performs a bi-level local search: an inner loop computes a sequence of Hamiltonian paths by successive minimum cost edge-(or arc-)exchanges; each time an edge is exchanged, it is fixed and cannot be exchanged again. The loop terminates when no more edges can be exchanged. At each iteration of the inner loop a Hamiltonian cycle is also computed (simply connecting the endpoints of the current Hamiltonian path). Therefore a sequence of feasible solutions is also produced; the best of them is the starting point for a new local search step (outer loop). The outer loop is terminated when no improvement is found.
The same idea can be applied whenever two subroutines are available to compute a quasi-feasible solution from another one and to compute a feasible solution from a quasi-feasible one. A quasi-feasible solution of the TSP is a Hamiltonian path; in the case of the VRPPD a quasi-feasible solution can be defined, for instance, as a solution obtained by the deletion of a customer visit from a tour. From any quasi-feasible solution another one can be obtained by a minimum cost (strong or weak) exchange of two visits; a feasible solution can be obtained from any quasi-feasible one, simply inserting the missing visit in the minimum cost feasible position. The resulting algorithm combines the features of EXCHANGE (strong or weak) and RELOCATE in a bi-level local search. The application of the same idea to CROSS is a straightforward extension of the Lin-Kernighan algorithm for the ATSP, where local search is performed just through arc exchanges. Further experimentation is needed to evaluate the effectiveness of this approach.

Visits grouping and splitting. A variation of the local search algorithm is the following. If a local search step produces a tour in which the same customer is visited twice consecutively, then the two visits are grouped into one and the resulting visit is no more divisible. This variation constrains the search and reduces the number of possible exchanges. In fact the observed quality of the solutions obtained in this way is slightly worse. On the contrary, better results can be achieved if the local search algorithm is allowed to split up the visits even more than the constructive algorithm does. This suggests new neighborhood definitions. Consider for instance the following variation of the RELOCATE neighborhood. Let $g_{\hat{t},k}$ denote the savings obtained by the deletion of the visit $v_{\hat{t},k} = (i, r_i)$ from tour $\hat{t}$. For each tour t and for each position $k = 0, \ldots, C_t$ along it, c_{itk} indicates the cost of the insertion of a visit to customer i in position k along tour $t \neq \hat{t}$ and q_{itk} indicates the maximum amount of load that can be picked-up from or delivered to customer i (according to the sign of r_i) in that position. Consider the following optimization subproblem to be solved for each given visit (i, r_i) in each given tour $\hat{t}$ at each local search iteration.

$$\min \ z_{i\hat{t}} = \sum_{t=1,\ldots,V, \ t\neq\hat{t}} (\sum_{k=0}^{C_t} c_{itk} \ x_{tk})$$

subject to

$$\sum_{t=1,\ldots,V, \ t\neq\hat{t}} (\sum_{k=0}^{C_t} q_{itk} \ x_{tk}) \geq r_i$$

$$\sum_{k=0}^{C_t} x_{tk} \leq 1 \quad \forall t = 1, \ldots, V, \ t \neq \hat{t}$$

$$x_{tk} \ \in \{0, 1\}, \quad \forall t = 1, \ldots, V \ \ t \neq \hat{t}, \quad k = 0, \ldots, C_t$$

It requires to choose the most convenient set of insertion positions in order to satisfy the demand r_i with the additional constraint that no more than one insertion position must be chosen for each vehicle. This problem is quite similar to the Multiple-Choice Knapsack Problem (MCKP) [10] with the only difference that inequalities replace equalities in the last constraints set. Every solution in this RELOCATESPLIT neighborhood has a cost equal to $z^*_{i,\hat{t}} - g_{\hat{t},k}$. In this way one customer visit can be splitted up into several ones at each local search iteration. The MCKP is known to be NP-hard (it includes the Knapsack Problem as a particular case) but it is possible to solve it with fast approximation algorithms. For instance a greedy approach consists of sorting the feasible insertion positions according to some policy and to select them sequentially until all the demand is satisfied; each time a position is selected, all insertion positions in the same vehicle are discarded from further consideration. I performed some limited tests and observed that the performance of the RELOCATESPLIT algorithm is sometimes better and sometimes worse than that of RELOCATE. It is quite apparent that RELOCATESPLIT performance is strongly affected by the policy followed by the greedy algorithm for the MCKP. Therefore a good idea is to approximate the MCKP with two or three different greedy algorithms and to choose the best solution found. For instance, some complementary sorting policies are: by increasing values of insertion cost, by decreasing values of load that can be inserted, by increasing value of the ratio between cost and load. The same "visits splitting" idea can be applied as well to the other neighborhoods illustrated in this section.

5 Indivisible demands

In many problems involving mixed pick-ups and deliveries the loading and unloading operations are constrained for budgetary or technical reasons, so that each customer must be completely served by one vehicle. An intermediate case happens when each customer can be served by two vehicles, one for the pick-up and the other for the delivery. When demands are not divisible, the minimization of the number of vehicles becomes a difficult task in itself, since it implies the solution of a two-dimensional bin-packing problem, that is NP-hard [11]. In this case the minimization of the total lenght is usually an objective of secondary importance compared to the minimization of the number of vehicles. However, once computed the number of vehicles and an initial feasible solution, all local search algorithms described in section 4 can be used to minimize the tours length. When the number V of vehicles is given, a mixed-integer programming model of the VRPPD with indivisible pick-up an delivery demands p_i and d_i for each customer i is the following.

$$min \quad \sum_{i=0}^{N}\sum_{j=0}^{N} c_{ij}x_{ij}$$

subject to

$$\sum_{i=0}^{N} x_{ij} = 1 \quad \forall j = 1, \ldots, N$$

$$\sum_{j=0}^{N} x_{ij} = 1 \quad \forall i = 1, \ldots, N$$

$$\sum_{j=0}^{N} x_{0j} = V$$

$$\sum_{k=0}^{N} y_{jk} - \sum_{i=0}^{N} y_{ij} = \begin{cases} p_j & \text{if } j \neq 0 \\ -Q_p & \text{if } j = 0 \end{cases}$$

$$\sum_{i=0}^{N} z_{ij} - \sum_{k=0}^{N} z_{jk} = \begin{cases} d_j & \text{if } j \neq 0 \\ -Q_d & \text{if } j = 0 \end{cases}$$

$$y_{ij} + z_{ij} \leq Q x_{ij} \quad \forall i, j = 0, \ldots, N$$

$$x_{ij} \in \{0, 1\}, \quad y_{ij}, z_{ij} \geq 0 \quad \forall i, j = 0, \ldots, N$$

In this model N indicates the total number of customers and Q_p and Q_d the total quantities to be picked-up and delivered. The number of variables in this case is much smaller than in the model of the VRPPD with divisible demands presented in section 1, since the number N of customers is likely to be much smaller than the number M of 1-customers.

The constructive approximation algorithms illustrated in section 3 can be applied to the problem with indivisible demands too, but WEAKSKIP looses the property of finding the minimum feasible number of vehicles. Local search neighborhoods defined in section 4 can still be applied to this case (except RELOCATESPLIT).

On the contrary, if the minimum number of vehicles is unknown, the local search algorithms illustrated above can be used to compute a heuristic solution in which the number of vehicles is also approximately minimized: instead of solving a two-dimensional bin-packing, the minimization of the number of vehicles is pursued with the trick of adding a sufficiently large cost to all depot-to-customer distances, so that solutions using more vehicles are penalized; in a similar way multiple visits to the same customers can be penalized by adding suitable large costs to all distances between different customer locations.

The experimental evaluation of both constructive and local search algorithms for the VRPPD with indivisible demands is a subject for future research, and so is the development of exact optimization algorithms for both versions of the VRPPD.

Acknowledgements

I am grateful to prof. M.G. Speranza and to the organizers and speakers of the IV International Workshop on Distribution Logistics (Brescia, 1998), who brought reverse logistics problems to my attention. I acknowledge the kind support of the Associazione Cremasca Studi Universitari, that sponsored the Operations Research Laboratory in the university site of Crema, making this study possible.

References

[1] Altinkemer K., Gavish B., *Heuristics for delivery problems with constant error guarantees* Transportation Science 24 (1990), 294-297

[2] Anily S., Bramel J., *Approximation Algorithms for the Capacitated Traveling Salesman Problem with Pickups and Deliveries* Tristan III, Puerto Rico (1998)

[3] Anily S., Mosheiov G., *The traveling salesman problem with delivery and backhauls* Operations Research Letters 16 (1994), 11-18

[4] Fleischmann M., Bloemhof-Ruwaard J.M., Dekker R., van der Laan E., van Nunen J.A.E.E., van Wassenhove L.N., *Quantitative models for reverse logistics: a review* European Journal of Operational Research 103 (1997), 1-17

[5] Gendreau M., Laporte G., Potvin J., *Vehicle routing: modern heuristics* in Local Search in Combinatorial Optimization, Aarts E. and Lenstra J.K. eds., Wiley-Interscience Series in Discrete Mathematics and Optimization, Chichester 1997

[6] Gendreau M., Laporte G., Vigo D., *Heuristics for the Traveling salesman problem with pickup and delivery* Computers and Operations Research 26 (1999), 699-714

[7] Haimovic M., Rinnooy Kan A., *Bounds and heuristics for capacitated routing problems* Mathematics of Operations Research 10 (1985), 527-542

[8] Kindervater A.P., Savelsbergh M.W.P., *Vehicle routing: handling edge exchanges* in *Local Search in Combinatorial Optimization*, Aarts E. and Lenstra J.K. eds., Wiley-Interscience Series in Discrete Mathematics and Optimization, Chichester 1997

[9] Lin S., Kernighan B.W., *An effective heuristic algorithm for the traveling salesman problem* Operations Research 21 (1973), 498-516

[10] Martello S., Toth P., *Knapsack Problems, Algorithms and Computer Implementations* John Wiley & Sons, Chichester (1990)

[11] Martello S., Vigo D., *Exact solution of the two-dimensional finite Bin Packing Problem* Management Science 44 (1998), 388-399

[12] Mosheiov G., *The Travelling Salesman Problem with pick-up and delivery* European Journal of Operational Research 79 (1994), 299-310

[13] Mosheiov G., *Vehicle Routing with pick-up and delivery: tour-partitioning heuristics* Computer and Industrial Engineering 34, 3 (1998), 669-684

[14] Righini G., *Tecniche per aumentare la robustezza degli algoritmi iterativi per problemi di localizzazione multi-risorse* Technical Report 93-037, Dipartimento di Elettronica e Informazione, Politecnico di Milano, May 1993

[15] Righini G., *The Largest Insertion algorithm for the Traveling Salesman Problem*, Note del Polo - Ricerca 29, Polo Didattico e di Ricerca di Crema, Università di Milano, May 2000.

[16] Rosenkrantz D.J., Stearns R.E., Lewis II P.M., *An analysis of several heuristics for the traveling salesman problem* Siam Journal on Computing 6, 3 [563-581] 1977

[17] Brimberg J., Hansen P., Mladenovic N., Taillard E.D., *Improvements and comparison of heuristics for solving the multisource Weber problem* Technical Report IDSIA-33-97, June 1997

[18] Toth P., Vigo D., *A heuristic algorithm for the symmetric and asymmetric Vehicle Routing Problem with backhauls* European Journal of Operational Research 113 (1999), 528-543

[19] Toth P., Vigo D., *Exact algorithms for the Vehicle Routing Problem with backhauls* Transportation Science 31 (1997), 60-71

[20] Vigo D., *A heuristic algorithm for the asymmetric capacitated vehicle routing problem* European Journal of Operational Research 89 (1996), 108-126