Manuscript Number:

Title: A parallel adaptive metaheuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery

Article Type: Special Issue: Logistics

Keywords: Parallel Computing; Adaptive Metaheuristic; Vehicle Routing; Pickup and Delivery

Corresponding Author: Ph.D Student Anand Subramanian, M.Sc.

Corresponding Author's Institution: Universidade Federal Fluminense

First Author: Anand Subramanian, M.Sc.

Order of Authors: Anand Subramanian, M.Sc.; Lúcia A Drummond, Ph.D; Cristiana Bentes, Ph.D; Luiz S Ochi, Ph.D; Ricardo Farias, Ph.D

Abstract: This paper presents a parallel adaptive algorithm based on the Iterated Local Search (ILS) metaheuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery (VRPSPD). To the best for our knowledge this is the first parallel approach for this problem. The experiments were performed in a cluster with a multi-core architecture using up to 256 cores. The results obtained on the benchmark problems, available in the literature, show that the proposed algorithm not only improved several of the known solutions, but also presented a very satisfying scalability.

December 31, 2008

To
The Editors
Special Issue of Computers & Operations Research
Metaheuristics for Vehicle Routing
Industrial Systems Optimization Laboratory
University of Technology of Troyes, France

Dear Sirs,

Please find enclosed the article entitled "A parallel adaptive metaheuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery", authored by Anand Subramanian, Lúcia Drummond, Cristiana Bentes, Luiz Satoru Ochi and Ricardo Farias, for favor of publication in your esteemed Journal.

The article deals with a subject that is of particular interest in the context of the special issue "Metaheuristics for Logistics and Vehicle Routing". The authors further believe that the approach contained in the article is original and the results obtained are sufficiently relevant to merit publication.

The ABSTRACT follows:

This paper presents a parallel adaptive algorithm based on the Iterated Local Search (ILS) metaheuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery (VRPSPD). To the best for our knowledge this is the first parallel approach for this problem. The experiments were performed in a cluster with a multi-core architecture using up to 256 cores. The results obtained on the benchmark problems, available in the literature, show that the proposed algorithm not only improved several of the known solutions, but also presented a very satisfying scalability.

Manuscript CLASSIFICATION:

Computational methods.

Sincerely,

Anand Subramanian
Lúcia Drummond
Cristiana Bentes
Luiz Satoru Ochi
Ricardo Farias

A parallel adaptive metaheuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery

A. Subramanian[*,a], L.M.A. Drummond[a], C. Bentes[b], L.S. Ochi[a], R. Farias[c]

[a] *Universidade Federal Fluminense, Instituto de Computação, Rua Passo da Pátria, 156 - Bloco E, 3º andar, São Domingos, Niterói-RJ, 24210-240, Brasil*
[b] *Universidade Estadual do Rio de Janeiro, Departmento de Engenharia de Sistemas, Rua São Francisco Xavier, 524 - Bloco D, 5º andar, Rio de Janeiro-RJ, 20550-900, Brasil*
[c] *Universidade Federal do Rio de Janeiro, Centro de Tecnologia - Bloco H, sala 319, Cidade Universitária, Rio de Janeiro-RJ, 21945-970, Brasil*

Abstract

This paper presents a parallel adaptive algorithm based on the Iterated Local Search (ILS) metaheuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery (VRPSPD). To the best for our knowledge this is the first parallel approach for this problem. The experiments were performed in a cluster with a multi-core architecture using up to 256 cores. The results obtained on the benchmark problems, available in the literature, show that the proposed algorithm not only improved several of the known solutions, but also presented a very satisfying scalability.

Key words:
Parallel Computing, Adaptive Metaheuristic, Vehicle Routing, Pickup and Delivery

1. Introduction

The Vehicle Routing Problem with Pickup and Delivery (VRPPD) is one of the main classes of the Vehicle Routing Problem (VRP). Among the several VRPPD variants we can cite: VRP with Backhauls, VRP with mixed Pickup and Delivery; Dial-a-ride Problem; and the VRP with Simultaneous Pickup and Delivery (VRPSPD).

In this work, we deal, particularly, with the VRPSPD. This problem arises especially in the Reverse Logistics context. Companies are increasingly faced with the task of managing the reverse flow of finished goods or raw-materials.

[*]Corresponding author
Email addresses: `anand@ic.uff.br` (A. Subramanian), `lucia@ic.uff.br` (L.M.A. Drummond), `cristianabentes@gmail.com` (C. Bentes), `satoru@ic.uff.br` (L.S. Ochi), `rfarias@gmail.com` (R. Farias)

According to de Brito and Dekker [1] one can affirm that companies get involved with Reverse Logistics because they can profit from it (economics); or they are obliged (legislation); or they feel socially impelled (corporate citizenship).

The VRPSPD can be defined as follows. Let $G = (V, E)$ be a complete graph with a set of vertices $V = \{0, ..., n\}$, where the vertex 0 represents the depot ($V_0 = \{0\}$) and the remaining ones the clients. Each edge $(i, j) \in E$ has a non-negative cost c_{ij} and each client $i \in V - V_0$ has a demand q_i for delivery and p_i for pickup. Let $C = \{1, ..., v\}$ be the set of homogeneous vehicles with capacity Q. The VRPSPD consists in constructing a set up to v routes in such away that: (i) all the pickup and delivery demands are accomplished; (ii) the vehicle's capacity is not exceeded; (iii) a client is visited by only a single vehicle; (iv) the sum of costs is minimized.

Since the VRPSPD is a $\mathcal{NP}$-hard problem [2], heuristics methods have proved to be more suitable for dealing with such problem in terms of solution quality vs. computational cost. Even though there are very efficient heuristics for combinatorial optimization problems, they sometimes may have the drawbacks of lack of robustness and also consume considerable amount of time to obtain acceptable quality solutions for complex large-scale problems. Some ideas have been proposed to reduce these limitations, namely the incorporation of learning modules to increase the robustness and parallel versions to decrease the computational time without compromising the quality of the solutions.

In this paper, we present a parallel adaptive algorithm based on the sequential heuristic developed by Subramanian *et al.* [3]. The main features of the proposed approach are the automatic calibration of some parameters and the ability of exploring the high-level of parallelism inherent to recent multi-core clusters. The automatic calibration of parameters makes our algorithm auto-adaptive, avoiding the need of manual tuning. The ability of scaling to a large number of processors allows our algorithm to take advantage of the emerging technology of multi-core architectures. These architectures are an attractive option in high performance computing, as the aggregation of a significant number of processors stimulate the construction of "superclusters" with thousands of cores. The high-degree of parallelism of our algorithm enables us to explore these kind of clusters in order to solve large-size instances of the VRPSPD.

Our parallel algorithm was evaluated on a multi-core cluster with up to 256 cores. Our interests were to evaluate not only the quality of the solutions, but also the performance of the algorithm. In terms of the quality of the solutions, we were capable of improving the best known solution for several benchmark problems proposed in the literature. In terms of performance, we evaluated the speedup and the communication overhead. The results show that our parallel algorithm can take advantage of the increasing number of processors, especially when solving large-size instances. The communication overhead, that is usually a bottleneck in the execution of many applications running on high performance clusters, stayed bellow 15% of the execution time for large-size instances when the number of iterations, to be computed, is more than 8 times the number of processors.

The remainder of this paper is organized as follows. Section 2 enumerates some works related to the VRPSPD as well as those that employed parallel strategies to VRPs. In Section 3 we explain the proposed parallel algorithm. Section 4 contains the experimental results and a comparison with the ones reported in the literature. Section 5 presents the concluding remarks.

2. Related Works

The main contributions and/or approaches with respect to the VRPSPD are described in Table 1. One can verify that the number of publications regarding the VRPSPD has considerably grown since the seminal work of Min [4].

Table 1: VRPSPD related works

Work	**Year**	**Contributions and/or Approach**
Min [4]	1989	First Work Case study in a public library
Halse [5]	1992	Cluster-first, route-second strategy 3-opt procedure
Salhi and Nagy [6]	1999	Insertion based heuristics
Dethloff [2]	2001	Constructive heuristic based on cheapest insertion, radial surcharge and residual capacity
Angelelli and Mansini[7]	2001	Branch-and-price for the VRPSPD with TW
Vural [8]	2002	Genetic Algorithm
Gokçe [9]	2003	Ant Colony
Ropke and Pisinger [10]	2004	Large Neighborhood Search
Nagy and Salhi [11]	2005	Heuristics with different levels of feasibility
Crispim and Brandao [12]	2005	TS + VND
Dell'amico *et al.* [13]	2005	Branch-and-price based on dynamic programming and state space relaxation
Chen and Wu [14]	2006	Record-to-record travel + Tabu Lists
Montané and Galvão [15]	2006	TS Algorithm
Gribkovskaia *et al.* [16]	2006	TS for the VRPSPD with a single vehicle
Bianchessi and Righini [17]	2007	Constructive and Local Search Heuristics TS + VNS
Wassan *et al.* [18]	2008	Reactive TS
Subramanian and Cabral [19]	2008	ILS Heuristic for the VRPSPD with Time Limit
Subramanian *et al.* [3]	2008	ILS-VND Heuristic
Zachariadis *et al.* [20]	2009	TS + Guided Local Search

It is possible to observe from Table 1 that the heuristic approaches are by far the most popular. In the last half decade, different metaheuristics such as Tabu Search (TS), Genetic Algorithm (GA), Ant Colony (AC), Simulated Annealing (SA), Variable Neighborhood Descend (VND), Variable Neighborhood Search (VNS) and Iterated Local Search (ILS) have been applied to solve the VRPSPD. The TS based heuristics proposed in [18] and [20] and the ILS based heuristic presented in [3] have provided the best known solutions in most of the benchmark problems available in the VRPSPD literature.

A considerable number of works related to parallel strategies for the VRP have been developed in the last ten years. Due to the nature of the problem, many of them are often inspired in metaheuristics. Table 2 lists some of these applications that include the classical VRP, VRPPD, VRP with Time Windows (VRPTW), VRP with heterogeneous fleet (HVRP), periodical VRP (PVRP), Dynamic VRP (DVRP) and VRP with time limit (VRPTL). The reader is referred to the recent survey performed by Crainic [21] for further details concerning the parallel strategies employed.

Table 2: Parallel Metaheursitcs for the VRP

Work	**Year**	**Contributions and/or Approach**
Rochat and Taillard [22]	1995	Adaptive memory-based TS for the VRPTW
Rego and Roucairol [23]	1996	TS for the VRP
Ochi *et al.* [24]	1998	GA based on the island model for the HVRP
Gendreau *et al.* [25]	1999	Cooperative multi-thread for the dispatching DVRP
Schulze and Fahle [26]	1999	TS for the VRPTW
Gendreau *et al.* [27]	2001	Master-worker TS for the DVRP
Drummond *et al.* [28]	2001	GA based on the island model for the PVRP
Jozefowiez *et al.* [29]	2002	GA based on the island model for the VRP with balanced route lenghts
Czech and Czarnas [30]	2002	Master-worker SA for the VRPTW
Gehring and Homberger [31]	2002	Evolutionary Alg. and TS for the VRPTW
Caricato *et al.* [32]	2003	TS for the mixed VRPPD
Alba and Dorronsoro. [33]	2004	Fine-grained Cellular GA for the VRPTL
Doerner *et al.* [34]	2004	AC for the VRP
Berger and Berkaoui [35]	2004	Master-worker GA for the VRPTW
Attanasio *et al.* [36]	2004	Multi-thread TS for the Dial-a-ride Problem
Le Bouthillier and Crainic [37]	2005	Cooperative Evolutionary and TS algorithms for the VRPTW
Polacek *et al.* [38]	2006	VNS for the VRPTW with multi-depot

As stated in Table 2, in spite of the considerable number of parallel metaheuristcs proposed we are not aware of any parallel ILS algorithm for VRPs. In addition, there are only few works that make use of parallel approaches for solving variants of the VRPPD (see [32], [36], [25]).

3. Parallel Adaptive Metaheuristic

This section presents the proposed parallel adaptive approach. The parallel heuristic is embedded with the algorithm proposed by Subramanian *et al.* [3], which is based on the combination of the ILS and VND approaches.

3.1. Sequential heuristic

This subsection describes the sequential ILS-VND heuristic developed in [3]. Its main steps are summarized in the Alg. 1. The multi-start heuristic executes $MaxIter$ iterations (lines 4-22), where at each iteration a solution is generated by means of a greedy procedure (line 5). The main ILS loop (lines 8-17) seeks to improve this initial solution using a VND procedure (line 9) in the local search phase (intensification) as well as a set of perturbation mechanisms (line 15) in the diversification phase. The parameter $MaxIterILS$ represents the maximum number of perturbations allowed without improvements.

Algorithm 1 ILS-VND

```
1: Procedure ILS-VND($MaxIter, MaxIterILS$, $\gamma$, $v$)
2: LoadData( );
3: $f^* := \infty$;
4: **for** $k$:=1,..., $MaxIter$ **do**
5:    $s$ := GenerateInitialSolution($\gamma$, $v$, seed);
6:    $s^{'} := s$;
7:    $iterILS := 0$;
8:    **while** $iterILS \leq MaxIterILS$ **do**
9:       $s$ := VND($N(.)$, $f(.)$, $r$, $s$) {$r$ = # of neighborhoods}
10:      **if** $f(s) < f(s^{'})$ **then**
11:         $s^{'} := s$;
12:         $f(s^{'}) := f(s)$;
13:         $iterILS := 0$;
14:      **end if**;
15:      $s$ := Perturb($s^{'}$);
16:      $iterILS := iterILS + 1$;
17:   **end while**;
18:   **if** $f(s^{'}) < f^*$ **then**
19:      $s^* := s^{'}$;
20:      $f^* := f(s^{'})$;
21:   **end if**;
22: **end for**;
23: return $s^*$;
24: end ILS-VND.
```

3.1.1. Constructive Procedure

Let v be the number of vehicles (or routes). Each route is filled with a client e, randomly selected from a Candidate List (CL). Next, the remaining clients are evaluated according to eq. (1). Note that every time a client is inserted to the partial solution, the CL must be updated and the insertion costs of the clients re-evaluated. The greedy insertion strategy here adopted was inspired in [2].

$$g\left(e\right) = \left(c_{ik} + c_{kj} - c_{ij}\right) - \gamma\left(c_{0k} + c_{k0}\right) \tag{1}$$

Function $g(e)$ presented in eq. (1) denotes the insertion cost of the client $e \in \text{CL}$ in a given route. The first parcel computes the insertion cost of the client k between the adjacent clients i and j. The second parcel corresponds to a surcharge used to avoid late insertions of clients located far away from the depot. The cost back and forth from the depot is weighted by a factor γ. The client e that has an insertion cost which leads to the minimum value of g is then added to the solution. The constructive procedure ends when all the clients have been added to the solution.

3.1.2. Local Search

A VND based algorithm [39] is used in the local search phase. This heuristic systematically modifies the neighborhood structures that belong to a set N in a deterministic way.

A set of ten neighborhood structures was utilized in an exhaustive fashion. Just the feasible movements are admitted, i.e., the ones that do not violate the maximum load constraints. Therefore, every time an improvement occurs, one should check whether this new solution is feasible or not among the ten neighborhoods adopted, six perform movements between different routes and four inside the routes. The N set of neighborhoods (between the routes) is described next. The other four are presented later in this subsection.

Shift(1,0) – $N^{(1)}$ – A client c is transferred from a route r_1 to a route r_2. The vehicle load is checked as follows. All clients located before the insertion's position have their loads added by q_c (delivery demand of the client c), while the ones located after have their loads added by p_c (pick-up demand of the client c). It is worth mentioning that certain devices to avoid unnecessary infeasible movements can be employed. For instance, before checking the insertion of c in some certain route, a preliminary verification is performed in r_2 to evaluate the vehicle load before leaving, $\sum_{i \in r_2} q_i + q_c$, and when arriving, $\sum_{i \in r_2} p_i + p_c$, the depot. If the load exceeds the vehicle capacity Q, then all the remaining possibilities of inserting c in this route will be always violated.

Crossover – $N^{(2)}$ – The arc between adjacent clients c_1 and c_2, belonging to a route r_1, and the one between c_3 and c_4, from a route r_2, are both removed. Later, an arc is inserted connecting c_1 and c_4 and another is inserted linking c_3 and c_2. The procedure for testing the vehicle load is more complex in comparison to Shift(1,0). At first, the initial (l_0) and final (l_f) vehicle loads of both routes are calculated. If the values of l_0 and l_f do not exceed the vehicle capacity Q then the remaining loads are verified through the following expression: $l_i = l_{i-1} + p_i - q_i$. Hence, if l_i surpasses Q, the movement is infeasible.

Swap(1,1) – $N^{(3)}$ – Permutation between a client c_1 from a route r_1 and a client c_2, from a route r_2. The loads of the vehicles of both routes are examined in the same way. For example, in case of r_2, all clients situated before the position that c_2 was found (now replaced by c_1), have their values added by q_{c_1} and subtracted by q_{c_2}, while the load of the clients positioned after c_1 increases by p_{c_1} and decreases by p_{c_2}.

Shift(2,0) – $N^{(4)}$ – Two consecutive clients, c_1 and c_2, are transferred from a route r_1 to a route r_2. The vehicle load is tested as in Shift(1,0).

Swap(2,1) – $N^{(5)}$ – Permutation of two consecutive clients, c_1 and c_2, from a route r_1 by a client c_3 from a route r_2. The load is verified by means of an extension of the approach used in the neighborhoods Shift(1,0) and Swap(1,1).

Swap(2,2) – $N^{(6)}$ – Permutation between two consecutive clients, c_1 and c_2, from a route r_1 by another two consecutive c_3 and c_4, belonging to a route r_2. The load is checked just as Swap(1,1).

In case of improvement of the current solution, one should aim to further refine the quality of the routes that contributed to reduce the objective function, that is, those which participated in the last betterment move. Hence, the following different neighborhoods are explored:

Or-*opt* – One, two or three consecutive clients are removed and inserted in another position of the route.

2-*opt* – Two nonadjacent arcs are removed and another two are added to form a new route.

Exchange – Permutation between two clients.

Reverse – This movement reverses the route direction if the value of the maximum load of the corresponding route is reduced.

3.1.3. Perturbation Mechanism

A set P of three perturbation mechanisms were adopted. Whenever the Perturb() function is called, one of the movements described below is randomly selected.

Ejection Chain – $P^{(1)}$ –This perturbation works as follows. A client from a route r_1 is transferred to a route r_2, next, a client from r_2 is transferred to a route r_3 and so on. The movement ends when one client from the last route r_v is transferred to r_1. The clients are chosen at random and the movement is applied only when there are up to 12 routes. It has been noticed that the application of the Ejection Chain, in most cases, had no effect when there are more than 12 routes.

Double-Swap – $P^{(2)}$ – Two Swap(1,1) movements are performed in sequence randomly.

Double-Bridge – $P^{(3)}$ – Consists in cutting four edges of a given route and inserting another four. This movement is randomly applied in all routes. When there are more then 15 routes, each of them has a 1/3 probability of being perturbed. This prevents exaggerated perturbations which may lead to unpromising regions of the solution space. These values were empirically calibrated.

3.2. Parallel Heuristic

The parallel adaptive metaheuristic developed (P-ILS-VND) is based on the master-worker model. Its steps are described in Alg. 2.

Two learning modules are incorporated into the algorithm as a remedy to the limitation of the sequential heuristic concerning the determination of the number of vehicles and the factor γ. In [3] these parameters were manually calibrated and it was verified that they are highly dependent on each particular instance.

The load balancing strategy is crucial to the performance of P-ILS-VND, since the time required by a processor to execute a given iteration of the ILS-VND is not known a priori and may vary a lot. Our load balancing strategy is based on dynamic distribution of work by the master. The idea is that the master performs the administrative role while the workers assume the computational roles.

Each process has an identification number (rank) from 0 to $p-1$, where p is the total number of processes. Our parallel adaptive metaheuristic is divided in three main parts, where the first two are related to the automatic calibration of the two parameters mentioned above and the third part is associated with the optimization phase.

The first part of the algorithm computes the number of vehicles (lines 4-12). Let n be the number of clients. Each one of the p process starts with $n-1$ vehicles (line 4) and then the ILS-VND heuristic is applied aiming to reduce the number of necessary vehicles (line 5). Due to computational cost reasons, this version of the heuristic makes use of just a single perturbation without improvement ($MaxIterILS = 1$). Firstly, the constructive procedure (see Subsection 3.1) of this method enforces each route to receive a client. Next, the movements employed in the local search phase will automatically decrease the number of vehicles needed. The master process receives the number of vehicles found by each worker and broadcasts the smallest value (lines 6-12).

The second part is related to the calibration of the factor γ (lines 13-21). Each process updates the value of γ according to its rank number (line 13). These values are previously assigned to every rank and they belong to a range $[0, 2]$. A version with just pure local search, i.e, without perturbations ($MaxIterILS = 0$), of the ILS-VND heuristic is executed several times (line 14). The master process receives the average costs from each worker and broadcasts the three best values of γ (lines 15-21), that is, those ones associated with the best average costs obtained.

The third part of the algorithm deals with the optimization itself (lines 22-40). Every process chooses, at random, one of the three best values for the parameter γ (line 22). The master process executes a single iteration of the ILS-VND (line 24) with the maximum number of perturbations without improvements ($MMaxIterILS$) smaller than those assigned to the workers ($WMaxIerILS$) in line 36. This should avoid the bottleneck that might occur if the master performs the same amount of perturbation as the workers do, since the execution time of the heuristic greatly increases with $MaxIterILS$. This guarantees that the master aways ends its unique iteration before all workers. The master is also in charge of the load balancing by feeding the workers with iterations whenever there is a request from one worker (lines 23-39). The iteration request is sent along with the cost of the last iteration obtained by the

Algorithm 2 P-ILS-VND

```
Procedure P-ILS-VND($iter$, $MMaxIterILS$, $WMaxIterILS$, $p$)
LoadData( );
$\gamma := 0$;
$v := N - 1$; {$v$ = # of vehicles, $n$ = # of clients}
ILS-VND(1, 1, $v$, $\gamma$); {each process updates $v$}
**if** master **then**
   **receive** $v$ from each worker;
   **send** the smallest $v$ to all workers;
**else**
   **send** $v$ to the master;
   **receive** $v$ from the master;
**end if**;
$\gamma :=$ updategamma($rank$);
Execute ILS-VND(5, 0, $v$, $\gamma$) 15 times and compute the avg. cost;
**if** master **then**
   **receive** the avg. cost from each worker;
   **send** the 3 best values of $\gamma$ to all workers;
**else**
   **send** the avg. cost to the master;
   **receive** the 3 best values of $\gamma$ from the master;
**end if**;
choose a $\gamma$ at random from the 3 best values of $\gamma$;
**if** master **then**
   ILS-VND(1, $MMaxIterILS$, $v$, $\gamma$);
   **while** $iter > 0$ **do**
      **receive** an iteration request and the solution cost from any worker;
      update $bestcost$;
      **send** an iteration to the worker who has requested;
      $iter := iter - 1$;
   **end while**;
   **receive** an iteration request and the solution cost from each worker;
   update $bestcost$;
   **send** a msg to all workers informing that there is no more iterations;
**else**
   **while** $iter > 0$ **do**
      ILS-VND(1, $WMaxIterILS$, $v$, $\gamma$);
      **send** a msg containing the cost and a request for iteration;
      **receive** a msg informing whether there is iteration from the master;
   **end while**;
**end if**;
end P-ILS-VND.
```

respective worker (line 37). In case of improvement, the master updates the *bestcost* and either replies with an iteration (lines 27-28) or with a message informing that there is no more iterations available (lines 32-33). The parameter *iter* denotes the total number of iterations available in the master process. The algorithm terminates when the master has no more iterations to provide and all the workers are aware of it.

4. Computational Results

The experiments were performed on a cluster with 32 SMP nodes, where each node consists of 2 Intel Xeon 2.66 GHz quad-core processors that share a 16G RAM. The nodes are connected via Gigabit Ethernet network. All the 256 cores run Linux CentOS, and the communication is handled using the OpenMPI library [40].

The parallel algorithm was coded in C++ and it was tested on benchmark problems proposed by Dethloff [2], Salhi and Nagy [6] and Montané and Galvão [15]. In [2] there are 40 instances with 50 clients; in [6] there are 14 instances involving 50 to 200 clients; and in [15] there are 18 instances involving 100 to 400 clients.

Three main points are tackled in this section, namely: quality of the solutions, performance evaluation (speedup) and communication overhead.

4.1. Quality of the Solutions

In this section we analyze the quality of the solutions generated by P-ILS-VND when 128 and 256 cores are used for the computation. The maximum number of perturbations without improvement adopted for the workers ($WMaxIterILS$) was 300. For the master, we have fixed $MMaxIterILS = 50$. The number of iterations to be computed ($iter$), available in the master, varies according to the number of processes. For these experiments, we used $iter = 2 \times (p - 1)$. We have executed the algorithm 50 times for each instance.

Although we are not directly interested in the performance of the algorithm in this subsection, one can observe in Tables 3, 4 and 6 that the average execution time (Avg. t) in all instances was higher when 256 cores were considered. This was expected since in the number of iterations to be computed increase with the number of processors used.

Table 3 shows the results obtained in Dethloff's instances. For all cases the P-ILS-VND, with both 128 and 256 cores, have equaled the best result found in the literature. It is relevant to mention that except for the instances SCA3-0, SCA8-2, SCA8-7 and CON3-2 the average solution (Avg. Sol.) of each instance coincides with the best solution (Best Sol.). The average gap between the Avg. Sol. and the literature solution both for 128 and 256 cores was only 0.01%.

The results obtained in Salhi and Nagy's 14 instances are presented in Table 4. When executed in 128 cores, the P-ILS-VND improved 3 results and equaled another 3; while in 256 cores the best known solutions were improved in 3 instnaces and equaled in another 4. A comparison just between these versions illustrates that the Best Sols. and Avg. Sols. were always better when 256

Table 3: Results obtained in Dethloff's instances

Instance	Clients	v	Literature	P-ILS-VND (128 cores)			P-ILS-VND (256 cores)		
				Best Sol.	Avg. Sol.	Avg. t	Best Sol.	Avg. Sol.	Avg. t
SCA3-0	50	4	**635.62**	**635.62**	636.05	1.99	**635.62**	636.03	4.92
SCA3-1	50	4	**697.84**	**697.84**	697.84	1.93	**697.84**	697.84	4.94
SCA3-2	50	4	**659.34**	**659.34**	659.34	1.78	**659.34**	659.34	4.82
SCA3-3	50	4	**680.04**	**680.04**	680.04	2.03	**680.04**	680.04	5.13
SCA3-4	50	4	**690.50**	**690.50**	690.50	1.76	**690.50**	690.50	4.69
SCA3-5	50	4	**659.90**	**659.90**	659.90	2.02	**659.90**	659.90	5.06
SCA3-6	50	4	**651.09**	**651.09**	651.09	1.79	**651.09**	651.09	4.72
SCA3-7	50	4	**659.17**	**659.17**	659.17	1.76	**659.17**	659.17	4.68
SCA3-8	50	4	**719.47**	**719.47**	719.47	1.76	**719.47**	719.48	4.78
SCA3-9	50	4	**681.00**	**681.00**	681.00	1.83	**681.00**	681.00	4.79
SCA8-0	50	9	**961.50**	**961.50**	961.50	2.48	**961.50**	961.50	5.63
SCA8-1	50	9	**1049.65**	**1049.65**	1049.65	2.52	**1049.65**	1049.65	5.45
SCA8-2	50	9	**1039.64**	**1039.64**	1040.48	2.40	**1039.64**	1039.75	5.53
SCA8-3	50	9	**983.34**	**983.34**	983.34	2.48	**983.34**	983.34	5.54
SCA8-4	50	9	**1065.49**	**1065.49**	1065.49	2.48	**1065.49**	1065.49	5.44
SCA8-5	50	9	**1027.08**	**1027.08**	1027.08	2.61	**1027.08**	1027.08	5.76
SCA8-6	50	9	**971.82**	**971.82**	971.82	3.21	**971.82**	971.82	6.23
SCA8-7	50	10	**1051.28**	**1051.28**	1052.97	3.26	**1051.28**	1052.33	6.21
SCA8-8	50	9	**1071.18**	**1071.18**	1071.18	2.46	**1071.18**	1071.18	5.53
SCA8-9	50	9	**1060.50**	**1060.50**	1060.50	2.37	**1060.50**	1060.50	5.26
CON3-0	50	4	**616.52**	**616.52**	616.52	2.62	**616.52**	616.52	5.71
CON3-1	50	4	**554.47**	**554.47**	554.47	2.00	**554.47**	554.47	5.11
CON3-2	50	4	**518.00**	**518.00**	518.38	1.91	**518.00**	518.45	4.82
CON3-3	50	4	**591.19**	**591.19**	591.19	1.99	**591.19**	591.19	5.04
CON3-4	50	4	**588.79**	**588.79**	588.79	2.05	**588.79**	588.79	5.05
CON3-5	50	4	**563.70**	**563.70**	563.70	2.18	**563.70**	563.70	5.21
CON3-6	50	4	**499.05**	**499.05**	499.05	2.36	**499.05**	499.05	5.29
CON3-7	50	4	**576.48**	**576.48**	576.48	2.29	**576.48**	576.48	5.24
CON3-8	50	4	**523.05**	**523.05**	523.05	2.06	**523.05**	523.05	5.13
CON3-9	50	4	**578.25**	**578.25**	578.25	1.80	**578.25**	578.25	4.86
CON8-0	50	9	**857.17**	**857.17**	857.17	2.83	**857.17**	857.17	6.03
CON8-1	50	9	**740.85**	**740.85**	740.85	2.72	**740.85**	740.85	5.89
CON8-2	50	9	**712.89**	**712.89**	712.89	2.72	**712.89**	712.89	5.82
CON8-3	50	9	**811.07**	**811.07**	811.07	3.10	**811.07**	811.07	8.33
CON8-4	50	9	**772.25**	**772.25**	772.25	3.33	**772.25**	772.25	6.35
CON8-5	50	9	**754.88**	**754.88**	754.88	3.85	**754.88**	754.88	6.98
CON8-6	50	9	**678.92**	**678.92**	678.92	3.31	**678.92**	678.92	6.53
CON8-7	50	9	**811.96**	**811.96**	811.96	3.20	**811.96**	811.96	6.14
CON8-8	50	9	**767.53**	**767.53**	767.53	3.03	**767.53**	767.53	6.01
CON8-9	50	9	**809.00**	**809.00**	809.00	2.85	**809.00**	809.00	5.79

Table 4: Results obtained in Salhi and Nagy's instances

Instance	Clients	v	Literature	P-ILS-VND (128 cores)			P-ILS-VND (256 cores)		
				Best Sol.	Avg. Sol.	Avg. t	Best Sol.	Avg. Sol.	Avg. t
CMT1X	50	3	**466.77**	**466.77**	466.77	1.89	**466.77**	466.77	4.90
CMT1Y	50	3	**466.77**	**466.77**	466.77	1.95	**466.77**	466.77	5.03
CMT2X	75	6	**668.77**	684.21	684.62	5.56	684.21	684.58	8.43
CMT2Y	75	6	**663.25**	684.21	684.63	5.50	684.21	684.58	8.62
CMT3X	100	5	**721.27**	721.40	722.65	8.11	721.40	722.11	10.37
CMT3Y	100	5	**721.27**	721.40	722.71	8.09	**721.27**	721.97	11.20
CMT12X	100	5	**644.70**	662.22	663.13	7.61	662.22	662.76	10.98
CMT12Y	100	5*	**659.52**	662.22	663.56	7.93	662.22	662.81	11.03
CMT11X	120	4	838.66	835.81	851.25	14.14	**833.92**	848.46	17.60
CMT11Y	120	4	**830.39**	835.26	848.82	13.79	835.81	847.10	16.08
CMT4X	150	7	**852.46**	**852.46**	853.85	28.05	**852.46**	853.61	38.08
CMT4Y	150	7	**852.35**	852.46	854.00	27.28	852.46	853.43	30.50
CMT5X	199	10	1030.55	1029.44	1034.27	57.45	**1029.25**	1032.59	57.89
CMT5Y	199	10	1030.55	1029.44	1033.46	54.03	**1029.25**	1032.31	60.55

(*) The best known solution was found with 6 vehicles.

Table 5: Comparison between P-ILS-VND (256 cores) and literature results in Salhi and Nagy's instances

Instance	CW [14]		W [18]		Z [20]		S [3]		P-ILS-VND	
	Sol.	v	Sol.	v	Sol.	v	Sol.	v	Sol.	v
CMT1X	478.52	3	468.30	3	469.80	3	**466.77**	3	**466.77**	3
CMT1Y	480.78	3	458.96*	3	469.80	3	**466.77**	3	**466.77**	3
CMT2X	688.51	6	**668.77**	6	684.21	6	684.21	6	684.21	6
CMT2Y	679.44	6	**663.25**	6	684.21	6	684.21	6	684.21	6
CMT3X	744.77	5	729.63	5	**721.27**	5	721.40	5	721.40	5
CMT3Y	723.88	5	745.46	5	**721.27**	5	721.40	5	**721.27**	5
CMT12X	678.46	6	**644.70**	5	662.22	5	662.22	5	662.22	5
CMT12Y	676.23	6	**659.52**	6	662.22	5	662.22	5	662.22	5
CMT11X	858.57	4	861.97	4	838.66	4	839.39	4	**833.92**	4
CMT11Y	859.77	5	**830.39**	4	837.08	4	841.88	4	835.81	4
CMT4X	887.00	7	876.50	7	**852.46**	7	852.83	7	**852.46**	7
CMT4Y	**852.35**	7	870.44	7	852.46	7	852.46	7	852.46	7
CMT5X	1089.22	10	1044.51	9	1030.55	10	1030.55	10	**1029.25**	10
CMT5Y	1084.27	10	1054.46	9	1030.55	10	1031.17	10	**1029.25**	10

(*) Infeasible solution.

cores were considered. The average gap between the Avg. Sols. and the ones found in the literature was 1.01% and 0.92% for 128 and 256 cores respectively. The P-ILS-VND failed to improve/equal the literature solution of the instances CMT2X, CMT2Y, CMT3X, CMT12X, CMT12Y, CMT11Y and CMT4Y.

Table 5 shows a comparison between the P-ILS-VND and the algorithms that obtained the best results in Salhi and Nagy's instances, namely those proposed by Chen and Wu (CW) [14], Wassan et al (W). [18], Zachariadis et al (Z). [20] and Subramanian *et al.* (S) [3]. When individually comparing the P-ILS-VND with each one of these algorithms, one can verify that the P-IlS-VND produced, on average, superior results. It is important to point out that the result obtained by Wassan *et al.* [18] in the instance CMT1Y (458.96) is questionable. We have found the optimum solution of this instance (466.77) by means of the the mathematical formulation presented in [13]. Note that the optimum solution coincides with the solution found in [3] and by the P-ILS-VND.

In Montané and Galvão's instances (Table 6), the P-ILS-VND with either 128 or 256 cores improved the results of 12 out of 18 instances and equaled the other 6. The best solutions found by the version with 256 cores were slightly better than the one with 128. The average solutions follow the same behavior, but one can verify that the difference tends to increase with the number of clients. The average gap between the Avg. Sols. and those reported in the literature was -0.04% and -0.09% for 128 and 256 cores respectively. These results are quite impressive since it illustrates that even the Avg. Sols. found by the P-ILS-VND are, in most cases, better than the best known solutions.

4.2. Performance Evaluation

In this section we evaluate the performance of our parallel algorithm. We are interested in analyzing the speedup that P-ILS-VND achieves when more processors are used. In this experiment, we used a different approach for the workload distribution. Instead of increasing the number of iterations as the

Table 6: Results obtained in Montané and Galvão's instances

Instance	Clients	v	Literature	P-ILS-VND (128 cores)			P-ILS-VND (256 cores)		
				Best Sol.	Avg. Sol.	Avg. t	Best Sol.	Avg. Sol.	Avg. t
r101	100	12	1010.90	**1009.95**	1011.49	13.82	**1009.95**	1010.49	16.67
r201	100	3	**666.20**	**666.20**	666.20	15.52	**666.20**	666.20	18.19
c101	100	16	1220.26	**1220.18**	1220.89	9.76	**1220.18**	1220.88	13.12
c201	100	5	**662.07**	**662.07**	662.07	6.65	**662.07**	662.07	9.85
rc101	100	10	1059.32	**1059.32**	1059.32	9.77	**1059.32**	1059.32	12.98
rc201	100	3	**672.92**	**672.92**	672.92	5.59	**672.92**	672.92	8.82
r1_2_1	200	23	3371.29	3368.44	3382.54	51.90	**3360.07**	3378.01	58.34
r2_2_1	200	5	**1665.58**	**1665.58**	1665.72	34.91	**1665.58**	1665.62	37.69
c1_2_1	200	28	3640.20	**3628.62**	3637.56	90.95	3632.78	3637.43	98.22
c2_2_1	200	9	1728.14	**1726.59**	1728.25	45.60	**1726.59**	1728.18	50.66
rc1_2_1	200	23	3327.98	3312.62	3325.71	61.72	**3308.27**	3322.35	70.64
rc2_2_1	200	5	**1560.00**	**1560.00**	1560.00	33.36	**1560.00**	1560.00	36.72
r1_4_1	400	54	9695.77	**9613.09**	9653.69	586.33	9613.93	9645.36	612.04
r2_4_1	400	10	3574.86	3556.86	3575.53	246.32	**3554.01**	3572.55	265.33
c1_4_1	400	63	11124.29	11106.63	11123.75	786.96	**11102.00**	11119.08	842.31
c2_4_1	400	15	3575.63	3552.87	3568.32	258.27	**3549.14**	3563.99	265.08
rc1_4_1	400	52	9602.53	9529.42	9570.95	642.40	**9527.99**	9565.97	651.68
rc2_4_1	400	11	3416.61	3407.49	3416.43	269.25	**3403.70**	3410.27	293.34

number of processors increases, we keep the number of iterations fixed, and increase the number of processors to compute them. This way, we can evaluate the benefits of using more processors on the algorithm performance.

To perform this experiment, we used the instances sn1x, sn3x, r2_2_1, and r2_4_1, with $iter = 511$, $WMaxIterILS = 300$ and $MMaxIterILS = 50$. The instances used represent different problem sizes: sn1x (50 clients) and sn3x (100 clients) represent medium-size problems, and r2_2_1 (200 clients) and r2_4_1 (400 clients) represent large-size problems. The speedup computed is defined as the ratio between the time taken by the sequential code and that of the parallel implementation. Parallel and sequential elapsed times used for the speedup calculation are the average of 20 consecutive runs on a dedicated machine.

In Fig. 1 we show the speedups of P-ILS-VND for solving sn1x, sn3x, r2_2_1, and r2_4_1, on 2, 4, 8, 16, 32, 64, 128 and 256 cores in log scale. As we can observe in these figures, for up to 128 cores, P-ILS-VND achieves increasing speedups for all the four instances. For more than 128 cores, P-ILS-VND achieve increasing speedups only for the large instances. The medium instances do not provide sufficient work for more than 128 cores, and the communication overhead dominates the execution time, as will be seen in the next section. We can also observe that the speedups obtained for up to 32 cores are close the ideal. For more than 32 cores, however, the communication overhead slows down the computation, making our gains not so prominent. Nevertheless, at some point, increasing the number of cores is not profitable, since the number of iterations to compute is fixed.

When comparing the execution for medium and large instances, we find interesting results. The speedups are greater for medium instances. This is due to the greater probability of occurrances of load imbalance in the computation of the larger instances, for a large number of cores. The overall computational effort for each core depends upon the number of local searches performed, while

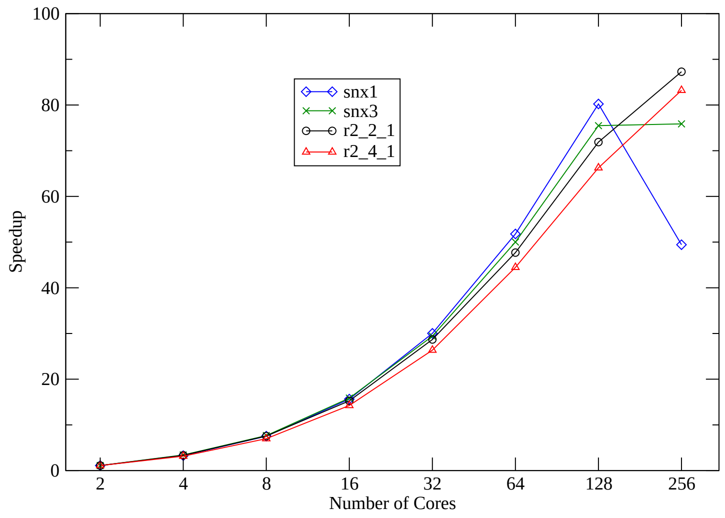

Figure 1: Speedup of P-ILS-VND for the four instances

the number of local searches performed varies randomly. The larger the instance, the greater the probability of one core having to compute more local searches than the other. Medium instances, on the other hand, tend to present faster convergence towards a good final solution, resulting in more homogeneous execution times for each core. This behavior can be observed by analyzing the standard deviation of the execution times obtained for each instance.

The speedup curves and the large number of processors involved in the computation show that our parallel implementation can successfully take advantage of the increasing number of processors in solving large-size problems. Furthermore, it shows that our parallel solution is scalable for big clusters.

4.3. Communication Overhead

Communication is one of the main factors that limit the speedup of a parallel application. In this section, we investigate the communication overhead generated by P-ILS-VND for different scenarios.

In P-ILS-VND, the communication occurs in the three computational phases. After the first phase, each process sends the computed number of vehicles to the master. The master, then, broadcasts the smallest number found. In the second phase, each process executes ILS-VND for different γ values and sends the average cost to the master. In the third phase, the master distributes the ILS-VND iterations to the workers.

The first and second phases are fast to compute and involve only one message from each worker to the master. The time spent in these phases, including computation and communication, is negligible in the overall execution time. The third phase is the one that really matters. The communication overhead in this phase depends on three parameters: the number of perturbations applied, the number of iterations to be computed, and the number of cores involved in the computation. The number of perturbations of the workers, $WMaxIterILS$, controls the granularity of the work performed in each iteration. The coarser the granularity (the greater number of perturbations), the greater the execution time of each iteration. The number of iterations to be computed controls the amount of ILS-VND iterations the master distributes to the workers. The number of iterations is defined by $iter = k \times (p - 1)$, where k is a parameter. The more iterations the algorithm computes, the greater the chance to find a best solution. The number of cores not only influences the number of iterations, but also the number of messages exchanged. As the number of core increases, more requests are made to the master.

In Tables 7, 8, and 9 we show the communication overhead and the number of messages generated by P-ILS-VND for the medium and large instances of the speedup experiment: sn1x, sn3x, r2_2_1 and r2_4_1. In these tables, we varied the number of cores, the parameter k (that influences the number of iterations), and the granularity of each iteration, $WMaxIterILS$. The communication overhead was computed as the percentage of the time the workers spend with communication, i.e., the percentage of the time spent on sending and receiving operations. We do not consider the master communication time in the communication overhead, as the master's main task is to communicate with the workers in order to distribute the work.

In Table 7 we show the communication overhead and the number of messages exchanged when k is fixed as 2, $WMaxIterILS$ is fixed as 300, and the number of cores goes from 16 to 256. The number of messages exchanged is the same for all the instances. As expected, with increase in the number of cores, the communication overhead and the number of messages also increase, since more workers are involved in the computation. We can also observe in this table that the medium instances have greater communication overhead. This occurs because they have less work to do, and so, the communication overhead becomes more prominent. For sn1x, when we used 256 cores, the communication overhead reaches more than 75% of the execution time. In this case, the communication overhead overrules the benefit of parallel processing, causing the slowdown in the speedup shown in the previous section.

In Table 8, we show the communication overhead and the number of messages exchanged for 256 cores, and $WMaxIterILS = 300$. We vary k from 2 to 12, generating scenarios where the number of iterations to be computed varies from 512 to 3072. As we can observe in these tables, as the value of k increases, the communication overhead decreases. These results are interesting, since we can also observe that the number of messages exchanged grows with the increase of k. The reason for the reduction in the communication overhead with the increase of k is that the load imbalance is reduced. The load imbalance has direct influence in the communication overhead, since the cores wait for the others to complete a receive operation. Higher values of k increase the number of iterations to be distributed, augmenting the chance for a good load balancing between the cores.

Table 7: Communication overhead and number of messages for $k = 2$ and $WMaxIterILS = 300$

# procs	Communication Overhead				# Messages
	sn1x	sn3x	r2_2_1	r2_4_1	
16	28%	28%	21%	21%	105
32	33%	33%	24%	26%	217
64	40%	36%	28%	30%	441
128	41%	40%	33%	31%	889
256	77%	56%	37%	35%	1785

Table 8: Communication overhead and number of messages for 256 processors and $WMaxIterILS = 300$

k	Communication Overhead				# Messages
	sn1x	sn3x	r2_2_1	r2_4_1	
2	77%	56%	37%	36%	1785
4	65%	41%	27%	25%	2295
6	56%	34%	21%	19%	2805
8	49%	27%	19%	17%	3315
10	44%	23%	15%	14%	3825
12	40%	20%	13%	12%	4335

In Table 9, we show the communication overhead for 256 cores, and $k = 2$. We vary $WMaxIterILS$ from 50 to 500, generating scenarios from finer to coarser granularity of work. The number of messages exchanged is not shown, since they do not change with the grouth of $WMaxIterILS$. As we can observe in this table, for large instances, as $WMaxIterILS$ increases, the communication overhead also increases. This is explained by the greater load imbalance generated. Each iteration to be computed present a different computational load, as a result of the randomness of the ILS-VND. Higher values of $WMaxIterILS$ produces iterations with even more different loads, hardening the load balancing task of the master. Medium instances, on the other hand, have fewer work for 256 cores, so the increase in $WMaxIterILS$ augments the amount of work to be done, reducing the weight of the communication in the overall execution time.

5. Concluding Remarks

In this paper, we have proposed a parallel adaptive metaheuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery. We believe it is the first parallel approach presented for this problem. The developed algorithm (P-ILS-VND) is based on the master-worker model and has three main parts. The first one estimates the number of vehicles; the second part corresponds to

Table 9: Communication overhead for 256 processors and $k = 2$

$WMaxIterILS$	Communication Overhead			
	sn1x	sn3x	r2_2_1	r2_4_1
50	87%	60%	29%	24%
100	86%	59%	33%	27%
200	81%	58%	36%	32%
300	77%	56%	37%	36%
400	74%	55%	37%	38%
500	72%	53%	39%	39%

the automatic calibration of the parameter γ, which is a factor that controls the bonus of inserting clients remotely located from the depot; and the third is the optimization phase.

The algorithm P-ILS-VND was tested in 72 benchmark problems with 50-400 clients and it was found capable of improving the results of 15 instances and equaling those of another 50. In terms of quality of the solutions, our heuristic has proven to be highly competitive, especially in large-size instances.

The high-degree of parallelism inherent in the P-ILS-VND allowed us to evaluate its performance on a cluster with up to 256 cores. In terms of speedup, we obtained increasing values for large-size instances (200-400 clients), showing the scalability of our algorithm for big clusters. We also evaluated the communication overhead generated by the algorithm for different number of processors; number of iterations; and granularity of the work. Our results show that for the medium instances (50-100 clients), when there are a great number of processors the communication overhead dominates the execution time. For larger instances, on the other hand, the communication overhead stays bellow 15% of the execution time when the number of iterations to be computed is more than 8 times the number of processors.

As future work, we intend to further improve P-ILS-VND to make better use of the hybrid communication environment provided by multi-core clusters, taking advantage of the shared memory on each computing node. We also intend to reduce the communication overhead in order to improve the scalability of our algorithm on bigger clusters.

References

[1] M. P. de Brito, R. Dekker, Reverse Logistics - Quantitative Models for Closed-Loop Supply Chains, Springer, 2004, Ch. Reverse Logistics: a framework, pp. 3–27.

[2] J. Dethloff, Vehicle routing and reverse logistics: the vehicle routing problem with simultaneous delivery and pick-up, OR Spektrum 23 (2001) 79–96.

[3] A. Subramanian, L. A. F. Cabral, L. S. Ochi, An efficient ILS heuristic for the vehicle routing problem with simultaneous pickup and delivery, Tech. rep., Universidade Federal Fluminense, available at http://www.ic.uff.br/˜satoru/index.php?id=2 (2008).

[4] H. Min, The multiple vehicle routing problem with simultaneous delivery and pick-up points, Transportation Research 23 (5) (1989) 377–386.

[5] K. Halse, Modeling and solving complex vehicle routing problems, Ph.D. thesis, Institute of Mathematical Statistics and Operations Research, Technical University of Denmark, Denmark (1992).

[6] S. Salhi, G. Nagy, A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling, Journal of the Operational Research Society 50 (1999) 1034–1042.

[7] E. Angelelli, R. Mansini, Quantitative Approaches to Distribution Logistics and Supply Chain Management, Springer, Berlin-Heidelberg, 2002, Ch. A branch-and-price algorithm for a simultaneous pick-up and delivery problem, pp. 249–267.

[8] A. V. Vural, A GA based meta-heuristic for capacited vehicle routing problem with simultaneous pick-up and deliveries, Master's thesis, Graduate School of Engineering and Natural Sciences, Sabanci University (2003).

[9] E. I. Gökçe, A revised ant colony system approach to vehicle routing problems, Master's thesis, Graduate School of Engineering and Natural Sciences, Sabanci University (2004).

[10] S. Röpke, D. Pisinger, A unified heuristic for a large class of vehicle routing problems with backhauls, Tech. Rep. 2004/14, University of Copenhagen (2004).

[11] G. Nagy, S. Salhi, Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries, European Journal of Operational Research 162 (2005) 126–141.

[12] J. Crispim, J. Brandao, Metaheuristics applied to mixed and simultaneous extensions of vehicle routing problems with backhauls, Journal of the Operational Research Society 56 (7) (2005) 1296–1302.

[13] M. Dell'Amico, G. Righini, M. Salanim, A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection, Transportation Science 40 (2) (2006) 235–247.

[14] J. F. Chen, T. H. Wu, Vehicle routing problem with simultaneous deliveries and pickups, Journal of the Operational Research Society 57 (5) (2005) 579–587.

[15] F. A. T. Montané, R. D. Galvão, A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service, European Journal of Operational Research 33 (3) (2006) 595–619.

[16] I. Gribkovskaia, Ø. Halskau, G. Laporte, M. Vlcek, General solutions to the single vehicle routing problem with pickups and deliveries, European Journal of Operational Research 180 (2007) 568–584.

[17] N. Bianchessi, G. Righini, Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery, Computers & Operations Research 34 (2) (2007) 578–594.

[18] N. A. Wassan, A. H. Wassan, G. Nagy, A reactive tabu search algorithm for the vehicle routing problem with simultaneous pickups and deliveries, Journal of Combinatorial Optimization 15 (4) (2008) 368–386.

[19] A. Subramanian, L. A. F. Cabral, An ILS based heuristic for the vehicle routing problem with simultaneous pickup and delivery, Proceedings of the Eighth European Conference on Evolutionary Computation in Combinatorial Optimisation, Lecture Notes in Computer Science 4972 (2008) 135–146.

[20] E. E. Zachariadis, C. D. Tarantilis, C. T. Kiranoudis, A hybrid metaheuristic algorithm for the vehicle routing problem with simultaneous delivery and pick-up service, Expert Systems with Applications 36 (2) (2009) 1070–1081.

[21] T. G. Crainic, The Vehicle Routing Problem: Latest advances and new challenges, Springer, 2008, Ch. Parallel Solution Methods for Vehicle Routing Problems, pp. 171–198.

[22] Y. Rochat, R. D. Taillard, Probabilistic diversification and intensification in local search for vehicle routing, Journal of Heuristics 1 (1995) 147–167.

[23] C. Rego, C. Roucairol, Meta-heuristics Theory and Applications, Kluwer, Dordrecht, 1996, Ch. A Parallel Tabu Search Algorithm Using Ejection Chains for the Vehicle Routing Problem, pp. 253–295.

[24] L. S. Ochi, D. S. Vianna, L. M. A. Drummond, A. O. Victor, A parallel evolutionary algorithm for the vehicle routing problem with heterogeneous fleet, Future Generation Computer Systems 14 (5-6) (1998) 285–292.

[25] M. Gendreau, F. Guertin, J.-Y. Potvin, E. Taillard, Parallel tabu search for real-time vehicle routing and dispatching, Transportation Science 33 (4) (1999) 381–390.

[26] J. Schulze, T. Fahle, A parallel algorithm for the vehicle routing problem with time window constraints, Annals of Operations Research 86.

[27] M. Gendreau, G. Laporte, F. Semet, A dynamic model and parallel tabu search heuristic for real-time ambulance relocation, Parallel Computing 27 (12) (2001) 1641–1653.

[28] L. M. A. Drummond, L. S. Ochi, D. S. Vianna, An asynchronous parallel metaheuristic for the period vehicle routing problem, Future Generation Computer Systems 17 (4) (2001) 379–386.

[29] N. Jozefowiez, F. Semet, E.-G. Talbi, Parallel and hybrid models for multi-objective optimization: Application to the vehicle routing problem, in: PPSN VII: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature, Springer-Verlag, London, UK, 2002, pp. 271–280.

[30] P. Czarnas, Parallel simulated annealing for the vehicle routing problem with time windows, in: 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, Canary Islands Spain, 2002, pp. 376–383.

[31] H. Gehring, J. Homberger, Parallelization of a two-phase metaheuristic for routing problems with time windows, Journal of Heuristics 8 (3) (2002) 251–276.

[32] P. Caricato, G. Ghiani, A. Grieco, E. Guerriero, Parallel tabu search for a pickup and delivery problem under track contention, Parallel Computing 29 (5) (2003) 631–639.

[33] E. Alba, B. Dorronsoro, Solving the vehicle routing problem by using cellular genetic algorithms., in: J. Gottlieb, G. R. Raidl (Eds.), EvoCOP, Vol. 3004 of Lecture Notes in Computer Science, Springer, 2004, pp. 11–20.

[34] K. Doerner, R. F. Hartl, G. Kiechle, M. Lucká, M. Reimann, Parallel ant systems for the capacitated vehicle routing problem, in: J. Gottlieb, G. R. Raidl (Eds.), EvoCOP, Vol. 3004 of Lecture Notes in Computer Science, Springer, 2004, pp. 72–83.

[35] J. Berger, M. Barkaoui, A parallel hybrid genetic algorithm for the vehicle routing problem with time windows, Computers & Operations Research 31 (12) (2004) 2037–2053.

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

[37] A. L. Bouthillier, T. G. Crainic, A cooperative parallel meta-heuristic for the vehicle routing problem with time windows, Computers & Operations Research 32 (7) (2005) 1685–1708.

[38] M. Polacek, S. Benkner, K. F. Doerner, R. F. Hartl, A cooperative and adaptive variable neighborhood search for the multi depot vehicle routing problem with time windows, BuR - Business Research 1 (2) (2008) 207–218.

[39] N. Mladenović, P. Hansen, Variable neighborhood search, Computers & Operations Research 24 (11) (1997) 1097–1100.

[40] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, T. S. Woodall, Open MPI: Goals, concept, and design of a next generation MPI implementation, in: Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, 2004, pp. 97–104.