Computers & Industrial Engineering 57 (2009) 277–286

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Iterated local search for the multiple depot vehicle scheduling problem

Benoît Laurent [a,b,*], Jin-Kao Hao [b]

[a] Perinfo SA, 41 Avenue Jean Jaurès, 67100 Strasbourg, France
[b] LERIA, Université d'Angers 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France

ARTICLE INFO

Article history:
Received 7 July 2008
Received in revised form 23 November 2008
Accepted 24 November 2008
Available online 16 December 2008

Keywords:
Multiple depot vehicle scheduling
Iterated local search
Constraints satisfaction

ABSTRACT

The multiple depot vehicle scheduling problem (MDVSP) is a well-known and important problem arising in public transport. Although many solution approaches have been published in the literature, algorithms using metaheuristics appeared only very recently (large neighborhood search and Tabu search). In this paper, we introduce an iterated local search algorithm for the MDVSP, incorporating a neighborhood schema called "block moves", based on the notion of ejection chains. Using a set of benchmark instances, we show empirically that the proposed algorithm performs better than the best metaheuristics implemented so far and obtains high quality results within short computational times.

1. Introduction

Given a set of trips and a set of vehicles housed in several depots of limited capacity, the multiple depot vehicle scheduling problem (MDVSP) aims at scheduling the vehicles to cover all the trips such that the resulting schedule satisfies a set of constraints and minimizes a cost function.

The MDVSP is a key step in the operational planning process of public transport companies. However, it is a challenging problem, shown to be NP-hard when two depots at least are considered (Bertossi, Carraresi, & Gallo, 1987).

The literature offers a range of solution methods developed in the last three decades. The early works on this problem focus on heuristic algorithms (see for instance Bodin, Golden, & Ball, 1983; Carraresi & Gallo, 1984; Rousseau, Lessard, & Désilets, 1988). Two main approaches coexisted essentially. The first one consists in clustering the trips and assigning them to the depot first, and then scheduling the vehicles in each depot separately. In the second approach, the whole fleet is first scheduled as if there were only one depot and then the resulting schedules are assigned to each depot. This latter idea is still employed in the construction of our initial solution.

Since the end of the 80s, several exact algorithms have been proposed. The models employed belong to one of the three following categories:

- single-commodity flow formulations (e.g. Carpaneto, Dell'Amico, Fischetti, & Toth, 1989; Fischetti, Lodi, Martello, & Toth, 2001),
- multicommodity flow formulations (e.g. Forbes, Holt, & Watts, 1994; Löebel, 1997; Löbel, 1998; Kliewer, Mellouli, & Suhl, 2006),
- set partitioning formulations (e.g. Ribeiro & Soumis, 1994; Hadjar, Marcotte, & Soumis, 2006; Oukil, Ben Amor, Desrosiers, & Gueddari, 2007).

Detailing the different models is beyond the scope of this paper. We refer interested readers to Odoni, Rousseau, and Wilson (1994) and Desaulniers and Hickman (2007) for a general presentation of transportation issues, including vehicle scheduling problems. For a dedicated and up-to-date survey on vehicle scheduling models, the reader is referred to Bunte, Kliewer, and Suhl (2006).

Integer linear programming remains undoubtedly the most popular approach to the MDVSP. Very recently and for the first time, two metaheuristic algorithms based on large neighborhood search (LNS) and Tabu search (TS) were reported (Pepin, Desaulniers, Hertz, & Huisman, 2008). Using a set of randomly generated instances, the authors compared these two novel approaches with three existing good performing heuristics: a heuristically applied CPLEX MIP solver, a Lagrangian heuristic and a column generation heuristic. They showed a dominance of the column generation heuristic in terms of solution quality. If a compromise between quality and computational time is aimed, LNS appears to be the best alternative, TS performing rather poorly. It should be noted that both the column generation heuristic and LNS metaheuristic rely on the GENCOL package developed and improved for more than 25 years (Desrosiers, Soumis, & Desrochers, 1984).

In this study, we present a new metaheuristic algorithm relying on iterated local search (ILS) to tackle the MDVSP. The proposed ILS algorithm has several original features. First, it is based on a

* Corresponding author. Address: Perinfo SA, 41 Avenue Jean Jaurès, 67100 Strasbourg, France. Tel.: +33 388449621; fax: +33 388449601.
E-mail addresses: blaurent@perinfo.com (B. Laurent), hao@info.univ-angers.fr (J.-K. Hao).

0360-8352/$ - see front matter © 2009 Published by Elsevier Ltd.
doi:10.1016/j.cie.2008.11.028

natural and high level model. Second, the ILS algorithm employs a powerful "block moves" neighborhood schema. Based on the idea of ejection chain, the "block moves" neighborhood aims at passing good properties of a solution on to its neighboring solutions. Third, the proposed algorithm employs an efficient auction algorithm (Freling, Wagelmans, & Paixão, 2001) for the generation of the initial schedule. Finally, it integrates a two-step perturbation mechanism, which provides the search with a controlled diversification facility.

We assess the performance of the proposed ILS algorithm on the set of the 30 MDVSP instances used in Pepin et al. (2008). Computational comparisons show that the ILS algorithm achieves very competitive results and constitutes a viable alternative to column generation based heuristics.

The remainder of this article is structured as follows. In Sections 2 and 3, we recall the MDVSP and present a formulation of the problem as a constraint satisfaction and optimization problem. In Section 4, the main components of the iterated local search are described. Section 5 is dedicated to computational experiments and comparison with previous results. The last section summarizes the main contributions of the work.

2. Multiple depot vehicle scheduling problem

In this section, we present the MDVSP which has the following input data:

- *Trips*: a set of n commercial trips, each one being characterized by an origin and a destination with associated starting and ending times; each trip needs to be covered by exactly one vehicle. Two commercial trips are said to be *feasible* if after serving the first trip, a vehicle has enough transfer time to begin the second trip. Without loss of generality, we assume that the trips are ordered by increasing starting time.
- *Vehicles and depots*: a fleet of p vehicles housed in m depots ($1 < m \ll n$) of limited capacity. A vehicle will be assigned a duty, i.e. a sequence of consecutively feasible trips. In real-world applications, there may exist requirements on the types of vehicles that can serve a commercial trip. Here, the fleet is supposed to be homogeneous such that trips can be performed by any vehicle.
- *Transfers without passengers*: before or after a commercial trip, a vehicle may perform a transfer without passengers. The transfer leaving a depot for reaching the first trip of the duty is called a pull-out trip. Its symmetrical counterpart is called a pull-in trip. The transfers between consecutive trips are called deadhead trips.

A valid vehicle schedule must satisfy the following four constraints:

(1) *Complete cover constraints*: all the trips must be covered by a vehicle.
(2) *Feasible sequence constraints*: two consecutive trips covered by a vehicle must be feasible.
(3) *Depot attachment constraints*: the vehicles return to the depot they start from.
(4) *Depot capacity constraints*: the number of vehicles used in each depot does not exceed the depot capacity.

Moreover, a valid vehicle schedule needs to minimize an objective function, composed of fixed and operational costs:

(1) *Fixed cost related to scheduled vehicles*: each scheduled vehicle represents a fixed and important cost P.
(2) *Operational costs*: these costs are induced by non-commercial trips, i.e. pull-in, pull-out and deadhead trips.

The cost structure indicates the importance accorded to the minimisation of the number of vehicles used while it leaves the operational costs as a secondary objective. For practical reason, P is spread over the costs of the pull-in and pull-out trips. Consequently, the multiple depot vehicle scheduling problem consists in determining a vehicle schedule that satisfies the set of imperative constraints (1)–(4) while minimizing the fixed and operational costs.

3. Problem formulation

In this section, a formulation of the MDVSP is proposed. For this purpose, we first introduce some basic notations.

- n, m, p: the number of trips, depots and vehicles.
- $\mathscr{T} = \{t_1, t_2, \ldots, t_n\}$: the set of n trips, ordered according to the increasing starting times.
- (s_i, e_i): the starting and ending times of a trip $t_i \in \mathscr{T}$.
- τ_{ij}, $(i,j) \in \{1,\ldots,n\}^2$, $i < j$: the transfer time from the end of trip t_i to the start of trip t_j.
- $\mathscr{H} = \{n+1, \ldots, n+m\}$: the set of m depots.
- r_h: the capacity of depot $h \in \mathscr{H}$.
- $\mathscr{V} = \{1, 2, \ldots, p\}$: the fleet of p vehicles.
- $\mathscr{V}_h \subset \mathscr{V}$: the subset of vehicles, housed in depot h. $\mathscr{V} = \mathscr{V}_1 \cup \ldots \cup \mathscr{V}_m \mathscr{V} = \{1, \ldots, r_{n+1}\} \cup \ldots \cup \{\sum_{k=1}^{m-1} r_{n+k} + 1, \ldots, p\}$
- c_{ij}, $(i,j) \in \{1,\ldots,n\}^2$, $i < j$: the cost of transfers without passengers. $c_{ij} = \infty$ if $i \geqslant j$ or if t_i and t_j are not feasible.
- c_{ki}, $(k,i) \in \mathscr{H} \times \{1,\ldots,n\}$: the cost of pull-out trips.
- c_{jk}, $(j,k) \in \{1,\ldots,n\} \times \mathscr{H}$: the cost of pull-in trips.
- $c_{hk} = 0$, $(h,k) \in \mathscr{H}^2$: there is no cost for transfers between depots.

3.1. Decision variables and solution representation

In the MDVSP, we aim at scheduling the vehicles to cover all the trips, i.e. assigning a vehicle to each trip. Thus, we naturally define the set of decision variables as the set of trips $\mathscr{T}$. Since each trip can be served by any vehicle, we use the set of vehicles $\mathscr{V}$ to define the value domain of each variable (trip) t_i.

Consequently, a candidate solution, i.e. a (feasible or infeasible) vehicle schedule designated by σ, is simply an assignment of a vehicle to each trip, which can be defined by a function σ: $\mathscr{T} \to \mathscr{V}$. Such a solution can be represented by a vector of length n, each element having a value from $\mathscr{V}$.

In our model, constraint (1) (trips coverage) is solved by construction. Moreover, a duty implicitly starts and ends at the depot of the vehicle it is assigned to, which satisfies constraint (3) (depot attachment). Finally, the depot capacities (constraint 4) are necessarily respected since each trip is assigned a vehicle belonging to one of the subset $\mathscr{V}_h$.

3.2. Constraints

As mentioned just previously, constraints (1), (3) and (4) are always satisfied in our model. Consequently, the only constraints to take into account in the solution procedure are constraints of type 2 (feasible sequence constraints) associated to the trips served by a vehicle. Now consider a given trip to vehicle schedule σ.

(1) *Complete cover constraints*: satisfied since σ is injective.
(2) *Feasible sequence constraints*: $\forall v \in \mathscr{V}, \forall (t_i, t_j) \in \mathscr{T}, i < j$,

$$(\sigma(t_i) = \sigma(t_j) = v) \Rightarrow e_i + \tau_{ij} \leqslant s_j$$

(3) *Depot attachment constraints*: satisfied by construction of the duties.
(4) *Depot capacity constraints*: satisfied by the definition of σ and injectivity.

3.3. Objectives

Given a trip to vehicle assignment σ, we introduce some binary variables x_{ij}, $(i,j) \in \{1,\ldots,n,n+1,\ldots,n+m\}^2$ in order to facilitate the computation of the objective function. The x_{ij} are deduced from σ and represent links between trips or between trips and depots. Different cases must be considered depending on the values of i and j:

- for $(i,j) \in \{1,\ldots,n\}^2$, $x_{ij} = 1$ if and only if trip t_j immediately follows trip t_i on the same vehicle; $x_{ij} = 0$ otherwise,
- for $i \in \mathscr{H}$ and $j \in \{1,\ldots,n\}$, $x_{ij} = 1$ indicates that j is the first trip of a vehicle in $\mathscr{H}$,
- for $i \in \{1,\ldots,n\}$ and $j \in \mathscr{H}$, $x_{ij} = 1$ means that i is the last trip of a vehicle in $\mathscr{H}$.
- for $(i,j) \in \mathscr{H}^2$, $x_{ij} = 0$.

A schedule is optimal if it minimizes the function given by

$$f(\sigma) = \sum_{(i,j)\in\{1,\ldots,n+m\}^2} c_{ij}x_{ij}$$

where c_{ij} represents the operational costs of the deadhead, pull-in and pull-out trips in σ (see notations at the beginning of Section 3). As indicated in Section 2, the fixed cost of each vehicle is integrated in its pull-in and pull-out trips.

4. Iterated local search

This section presents the iterated local search algorithm developed for the MDVSP. We first provide a general outline of the method and the framework on which the heuristic is applied, namely the search space and the evaluation function. The main ingredients of ILS are then detailed.

4.1. General ILS algorithm

Iterated local search is a neighborhood exploration paradigm that was formally defined in Lourenço, Martin, and Stützle (2002). The idea of ILS consists in exploring the space of local optima, with respect to a given neighborhood.

To devise an ILS algorithm, four components are needed: a *GenerateInitialSolution* procedure that generates an initial solution s_0, a procedure *Perturbation*, that modifies the current solution s^* leading to some intermediate solution s', a procedure *LocalSearch* that returns an improved solution s'', and an *AcceptanceCriterion* that decides to which solution the next perturbation is applied. A high level procedural view of ILS is sketched in Algorithm 1.

Algorithm 1. Pseudocode of an iterated local search procedure

```
s0 ← GenerateInitialSolution()
s* ← LocalSearch(s0)
repeat
  s' ← Perturbation(s*)
  s'' ← LocalSearch(s')
  s* ← AcceptanceCriterion(s*,s'')
until termination criterion met.
```

4.2. Search space and evaluation function

A candidate solution (or configuration) is an assignment of trips in $\mathscr{T}$ to vehicles in $\mathscr{V}$ which can be simply represented as a vector of integers.

$$\sigma = (\sigma(t_1), \sigma(t_2), \ldots, \sigma(t_n)) \quad (\sigma \in \mathscr{V}^n)$$

The search space composed of all the feasible and infeasible solutions can be defined by

$$\Sigma = \{\sigma | \mathscr{T} \to \mathscr{V}\}$$

As mentioned in Section 3.2, constraints (1), (3) and (4) are satisfied by construction. We thus define the *constrained (or feasible)* search space Ω comprising all the solutions of Σ that additionally satisfy constraint 2. The quality of a candidate solution $\sigma \in \Omega$ is assessed by means of the objective function f defined in Section 3.3.

In the next section, we show a procedure that generates initial solutions in Ω (as long as the problem is feasible). Then, the rest of the algorithm consistently maintains the exploration of the solutions within this subspace.

4.3. Initial solution

Like all neighborhood search algorithms, ILS needs an initial solution. This one is theoretically of little influence on long runs but its quality can be crucial when solutions of good quality are to be reached quickly (see Lourenço et al., 2002). For this reason, we devise a heuristic which is a slightly improved version of the algorithm introduced in Bodin et al. (1983), also employed in Pepin et al. (in press).

First, the MDVSP is transformed into a single-depot vehicle scheduling problem (SDVSP) by replacing all the depots $h \in \mathscr{H}$ by a single fictitious depot. We recall that the SDVSP is polynomial-time solvable. Costs for pull-in and pull-out trips in the relaxed formulation are equal to the cheapest costs among the different depots in the original problem. To solve the SDVSP, we implemented an efficient auction algorithm proposed in Freling et al. (2001). This first phase guarantees finding a minimum number of vehicles to cover all the trips in $\mathscr{T}$. It provides a set of duties that must be assigned a vehicle in the second phase.

The assignment of the duties is solved using a greedy procedure: each one is assigned a free vehicle yielding the least cost, regarding the costs of the pull-in and pull-out trips. We added a third phase in which a SDVSP is solved for each depot.

The complexity of the procedure mainly lies in the auction algorithm. The computational analysis performed in Schwarz (1994) indicates a complexity of $O(n^2 \log(n))$ for a forward auction algorithm for the linear assignment problem. However, our procedure integrates a combined version which alternates between forward and backward auctions (see Bertsekas, 1991). Computational results carried out in that paper indicate that the combined version is considerably faster than the forward version alone.

Applied to the instances of Pepin et al. (2008), the initial heuristic rapidly produces feasible solutions.

4.4. Local search and neighborhood structures

Our local search procedure performs a descent from the initial solution until it reaches a local minimum. Its strength lies in the employed neighborhood and in the way it is explored.

To show the effectiveness of the "block-moves" neighborhood, we first describe and analyze the two previous neighborhoods, $\mathscr{N}_{shift}$ and $\mathscr{N}_{swap}$, embedded in the Tabu search algorithm described in Pepin et al. (2008). Then, we define our "block-moves" mechanism which is based on the notion of ejection chains. In the rest of the paper, σ designates the current configuration, σ' a neighboring configuration taken from a given neighborhood.

4.4.1. Existing neighborhoods

In the shift neighborhood $\mathscr{N}_{shift}$, a neighboring configuration is obtained by transferring a trip of a vehicle to another vehicle. In

the swap neighborhood $\mathcal{N}_{swap}$, the underlying move implies two trips t_i and t_j, run by two different vehicles v and v'. It consists in exchanging the two vehicles of the two trips. The respective size of $\mathcal{N}_{shift}$ and $\mathcal{N}_{swap}$ is $O(np)$ and $O(n^2)$. When a change concerns the first or the last trip of a vehicle, one checks if a transfer of the entire sequence of trips to an available vehicle of another depot would be profitable.

4.4.2. Block-moves neighborhood

Recent studies have shown that compound neighborhood structures, which encompass successions of interdependent moves, have advantages over simple neighborhoods. In our case, we propose a neighborhood structure based on a form of ejection chains. This new neighborhood structure will allow us to explore the solution space more extensively and effectively.

The principle of ejection chain was introduced in Glover (1996) and defined in a very general way. An ejection chain is initiated by selecting a set of elements to undergo a change of state. The result of this change leads to identifying a collection of other sets, with the property that at least one of the elements must be "ejected from" their current state.

Our neighborhood mechanism, called *bl_moves*, consists in shifting *bl* consecutive trips run by the same vehicle v, to another vehicle v'. The generated compound moves are thus represented by a sequence of triplets (t, v, v'). The block to be shifted is randomly selected according to a uniform distribution. These ejection moves often cause constraints violations that will trigger repair attempts. For each conflicting trip previously handled by v', we scan the vehicles that may receive it and retain the best one regarding the objective function. Unresolved conflicts are not allowed. As a last resort, the trips involved in a conflict after the repairing step are assigned a free vehicle. If no free vehicle is available, the movement is abandoned. As in $\mathcal{N}_{shift}$ and $\mathcal{N}_{swap}$, complete transfers of the trips to vehicles belonging to other depots are evaluated. Such transfers imply triplets to be added to the set of atomic moves. Algorithm 2 describes the generation of a neighbor while Fig. 1 shows an illustrative example. From a given configuration (step 1), a block of two trips handled by vehicle 3 is randomly selected (step 2). Moving the block to vehicle 1 causes conflicts (steps 3 and 4). The trips involved in these conflicts are moved to vehicles 2 and 3 (steps 5 and 6).

The size of $\mathcal{N}_{bl\_moves}(\sigma)$ depends on the distribution of the trips over the vehicles in the current configuration σ. Let M_σ be the maximum number of trips handled by a vehicle in σ. Let nbv_σ be the function that associates to each block size *bl*, the number of vehicles running *bl* trips exactly. With these notations, the size of $\mathcal{N}_{bl\_moves}(\sigma)$ is computed in this way:

$$|\mathcal{N}_{bl\_moves}(\sigma)| = \left[\sum_{k=1}^{M_\sigma} \frac{k(k+1)}{2} \times nbv_\sigma(k)\right] \times (p-1)$$

The part of the formula between square-brackets deals with the number of blocks, each one having $p-1$ possible new values. Even if the size of $\mathcal{N}_{bl\_moves}(\sigma)$ varies during the search, this variation is rather limited.

Algorithm 2. Pseudocode for the random neighbor selection in $\mathcal{N}_{bl\_moves}(\sigma)$

Require: Configuration σ
Ensure: $\mathcal{M} \neq \emptyset$ {Set of atomic moves represented by triplets (t, v, v'), where t is a trip, v its vehicle in σ and v' its vehicle in the neighboring configuration σ'}
$\mathcal{M} \leftarrow \emptyset$
$(v, \mathcal{T}') \leftarrow$ rd_sel_bl(σ) {Select a sequence of trips handled by vehicle v}
$v' \leftarrow$ rand(p) {Select a new vehicle at random}
for all $t \in \mathcal{T}'$ **do**
  $\mathcal{M} \leftarrow \mathcal{M} \cup \{(t, v, v')\}$
end for
$\mathcal{T}'' \leftarrow$ detect_conflicts($\sigma, \mathcal{T}', v\prime$) {Already assigned trips to v' in conflict are added to $\mathcal{T}''$}
if $\mathcal{T}'' \neq \emptyset$ **then**
  for all $t \in \mathcal{T}''$ **do**
    $\mathcal{V}_{pot} \leftarrow$ potential(σ, t) {Set of vehicles that can handle trip t}
    $v_{min} \leftarrow$ best(σ, t) {Select the best new vehicle for t}
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(t, v\prime, v_{min})\}$
  end for
end if
{Check the pertinency of changes of depots: those are represented by atomic moves added to $\mathcal{M}$}
change_depots($\sigma, \mathcal{M}$)

The ideas underlying the definition of the *bl_move* neighborhood structure are the following. First, it prevents the search from being stuck in local optima because of some conflicts that could be repaired as soon as they arise. Second, behind the notion of "block moves", we aim at preserving the good properties of the configuration, typically the trips that fit well together.

4.4.3. Neighborhood sampling

In a random descent algorithm, one neighbor of the current configuration is randomly selected (with a uniform probability distribution or not) and evaluated at each step. The neighbor is accepted if its cost is smaller than or equal to the cost of the current solution. A drawback of the random descent is that potentially high quality neighbors can be definitively missed.

A steepest descent algorithm evaluates instead all the neighbors of the current configuration in order to select a best improving neighbor. Such a descent algorithm ensures a more intensive search at the price of higher computational efforts.

An intermediate technique consists in examining a fraction of the whole neighborhood of the current configuration. In our case, a predefined number of neighbors are randomly generated at each step. The best one in this subset is retained after evaluation. All the question then lies in the balance between quality and computational effort. In our algorithm, the neighborhood size is computed after the construction of the initial solution. A parameter, denoted α ($\alpha \in [0,1]$) is introduced to determine the portion of the whole neighborhood examined at each iteration.

4.4.4. Stopping criterion for local search

In the local search phase, when the number of iterations without strict improvement exceeds $n_{idle} = \frac{\beta}{\alpha} \times |\mathcal{N}_{bl\_moves}(\sigma)|$, the search is considered as stagnating on a local optimum and is subsequently halted. Note that n_{idle} takes into account the sample size.

4.5. Perturbation mechanisms

In order to escape from local optima and to explore new regions of the search space, ILS applies perturbations to the best local minimum. A crucial issue concerns the strength of the perturbation. If it is too strong, ILS may behave like a random restart resulting in a very low probability of finding better solutions. On the other hand, if the perturbation is not strong enough, the local search procedure will rapidly go back to a previous local optimum.

We defined two types of perturbation mechanisms that are sequentially applied in our ILS procedure. The first one consists in carrying out η moves independently of their effect on the eval-

Table 1
Average gaps to the best known solutions and standard deviation (%).

Depots	4						8					
Trips	500		1000		1500		500		1000		1500	
shift	13.62	2.99	10.86	3.49	11.98	2.64	20.10	1.86	19.19	3.88	21.95	6.57
swap	11.32	1.81	8.22	2.16	9.82	1.93	15.65	1.46	15.70	2.70	16.83	4.62
bl_moves	4.55	1.07	3.94	1.48	5.65	1.40	7.25	1.00	7.92	1.60	10.05	2.84

uation function as long as they do not increase the number of used vehicles. Thanks to the neighborhood sampling mechanism, we attain a double objective: achieving strong perturbations without deteriorating too much the configuration. The nature of the perturbation is also an important issue. If it is too weak, it will be undone during the local search phase. Our second perturbation mechanism performs a reorganization of the trips in each depot. More precisely, all the trips handled by the vehicles from one particular depot are first unassigned. Then, a SDVSP is solved to optimally reassign these trips. Being more complex than the block moves, such a perturbation is difficult to reverse by the local search phase. Furthermore, it optimizes some subparts of the configuration, resulting in better quality solutions.

4.6. Acceptance criterion

The acceptance criterion plays a role in the balance between intensification and diversification. Our acceptance criterion clearly privileges intensification since the best solution resulting from a local search phase is accepted if and only if it improves the local minimum encountered so far.

4.7. Termination criterion

Various conditions may be used to stop the algorithm: maximal number of iterations, lower bound, etc. Since our algorithm will be compared to other heuristics (see Section 5), the stopping criterion is the computational time here.

5. Computational experiments

5.1. Benchmarks and settings

The experiments that we carried out rely on the benchmarks[1] proposed in Pepin et al. (2008). These test problems aim at simulating real-world situations. The way they were generated has been originally described in Carpaneto et al. (1989) and recalled in Forbes et al. (1994, 2001). This set of benchmarks contains six categories ($m \in \{4,8\}$ and $n \in \{500,1000,1500\}$). Within each category, five instances are defined, leading to a total of 30 instances.

Our ILS algorithm was coded in C++, compiled with VC++ 8.0 (flag -O3), on a laptop equipped with a 2 GHz T7200 Intel Core and 2GB RAM running Windows XP.

In the rest of this section, the results often refer to the gap of a solution found by an algorithm to the best known solution (values available at the web site of the problem instances). As proposed in Carpaneto et al. (1989), the solution values are purged of the additional cost P for the vehicles (P is set to 10000), in order to get more discriminating gaps:

$$Gap = \frac{f(\sigma) - f(\sigma_{best})}{f(\sigma_{best}) - P \times nb_{vehi}(\sigma_{best})}$$

[1] Available for download at http://people.few.eur.nl/huisman/instances.htm.

5.2. Comparison of neighborhoods

The purpose of the first experiment is to compare $\mathcal{N}_{bl\_moves}$ with the two existing neighborhoods $\mathcal{N}_{shift}$ and $\mathcal{N}_{swap}$ on the 30 problem instances. For this purpose, we embed each of these neighborhoods into a random descent algorithm (see Section 4.4.3).

Due to their stochastic nature, the algorithms are executed 20 times on each instance. The stop criterion for each run is set to 5, 15 and 30 s for instances containing 500, 1000 and 1500 trips, respectively. For each problem instances, we first calculated the average gaps (see Section 5.1) of the 20 local optima found by the independent runs. The average is further realized on the five instances within a common instance type. Table 1 shows the gaps and the standard deviations obtained by the three descent algorithms for the six categories of problem instances.

From this table, it can be observed that the descent algorithm embedding $\mathcal{N}_{bl\_moves}$ obtains far better results than the two others. Standard deviations also show a greater robustness for our block-moves neighborhood $\mathcal{N}_{bl\_moves}$. For instance, for the category of 8 depots and 1000 trips, $\mathcal{N}_{bl\_moves}$ leads to a gap of 7.92 which is approximately the half of the gap obtained with $\mathcal{N}_{swap}$ and only one third of the the gap obtained with $\mathcal{N}_{shift}$. Moreover, the standard deviation of 1.60 is much smaller than those produced by the two other neighborhoods (2.70 and 3.88, respectively for $\mathcal{N}_{swap}$ and $\mathcal{N}_{shift}$). This experimentation also confirms that $\mathcal{N}_{swap}$ performs better than the simple shift-based neighborhood.

To complement the results of Table 1, we show in Fig. 2 the evolution of the cost function during the descent search process with the three neighborhoods. The curves represent the average values for each neighborhood over the 5 instances of 8 depots and 1500 trips. One clearly observes that the "block-moves" neighborhood allows the descent process to always reach better solutions from the first iterations to the end of the search.

Finally, to confirm the practical advantage of $\mathcal{N}_{bl\_moves}$ over $\mathcal{N}_{swap}$ and $\mathcal{N}_{shift}$, we replicated the same experiments with our ILS algorithm and observed quite similar results.

5.3. Quantitative and qualitative analysis

In order to have a deeper understanding of the behavior differences, we perform a qualitative and quantitative analysis of the three types of neighborhood. We focus on the evolution of two important indicators pertaining to neighborhoods during a search process: the number of improving and plateau neighbors and the quality (or strength) of improvement. For these investigations, we used an instance with 8 depots and 1000 trips (instance m8n1000s0), since it represents a test problem of reasonable size and difficulty.

In a preamble of these experimentations, the descent algorithm equipped with $\mathcal{N}_{bl\_moves}$ was executed 20 times (in order to make the analysis statistically sound). The configurations encountered when the evaluation function reaches predefined thresholds (from 24% to 8%, at every 2% step) were recorded for two post-analysis.

In the first experiment, we consider the previously collected configurations and compute the average number of improving, plateau and deteriorating neighbors. In Fig. 3, we represented the

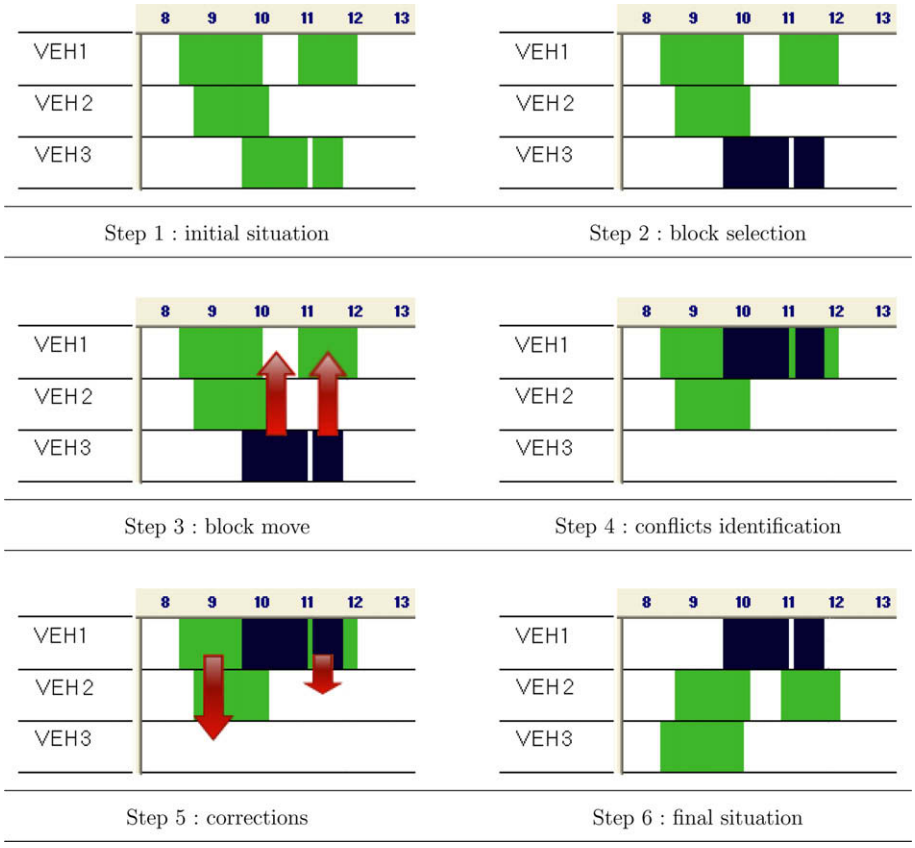

Fig. 1. Example of a block move with corrections.

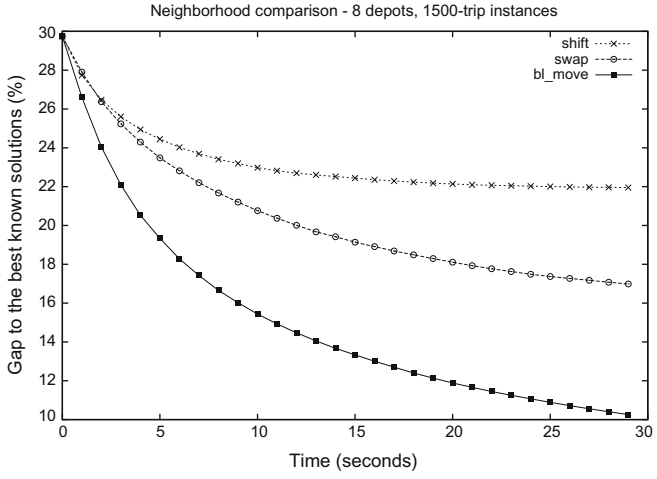

Fig. 2. Evolution of the cost function during the search according to the employed neighborhood.

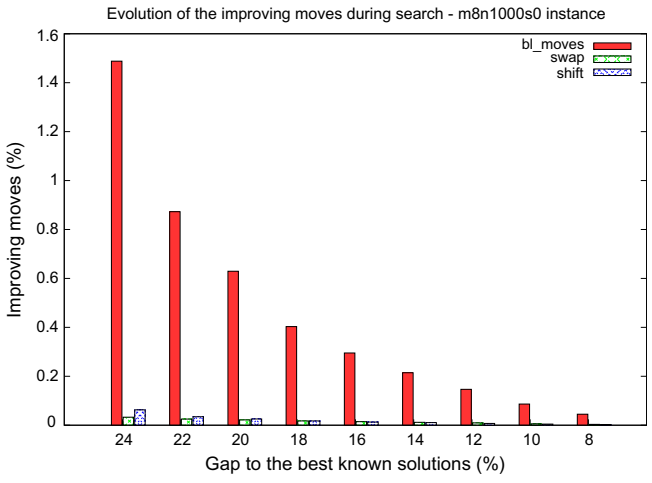

Fig. 3. Evolution of the percentage of improving neighbors during search – m8n1000s0 instance.

improving neighbors (in terms of percentage regarding the neighborhood size) according to the type of neighborhood. It is clearly observed that the "block-moves" neighborhood leads always higher percentages of improving neighbors during a descent, thus offering many more opportunities for improving the current solution at each iteration. Also notice that when improving neighbors begin to vanish with $\mathcal{N}_{shift}$ and $\mathcal{N}_{swap}$, the "block-moves" neighborhood $\mathcal{N}_{bl\_moves}$ continues to offer improving solutions. We can conclude that the descent with $\mathcal{N}_{bl\_moves}$ has much more chances to improve its solutions at each iteration and for a longer time than with $\mathcal{N}_{shift}$ and $\mathcal{N}_{swap}$.

In the second experiment, we examine another important factor of a neighborhood, namely the quality of the improvement, measured by the absolute cost difference $\Delta f = |f(\sigma') - f(\sigma)|$, σ being a solution and σ' an improving neighbor. Fig. 4 represents the results obtained on instance m8n1000s0 in form of boxplots for the three neighborhoods at the 8% threshold. The boxplots provide a convenient way for graphically depicting numerical data through their five-number summaries (the smallest observation, lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation).

The differences between medians (24.0 for *bl_moves*, 9.0 for *shift* and 20.0 for *swap*) already discriminate $\mathcal{N}_{shift}$ from the two other

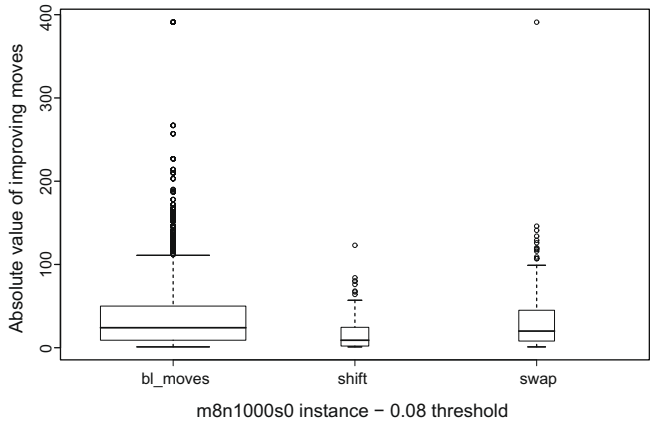

Fig. 4. Quality of improving neighbors according to the three neighborhoods – m8n1000s0 instance.

neighborhoods. Concerning $\mathcal{N}_{bl\_moves}$, we notice many outliers occurrences, which are observations numerically distant from the rest of the data. In the present case, these outliers correspond to very appealing movements. Improving neighbors appear not only more numerous in $\mathcal{N}_{bl\_moves}$ than in $\mathcal{N}_{shift}$ and $\mathcal{N}_{swap}$, but they also are of much better quality, permitting more important cost improvements.

This set of experiments leads us to conclude that the "block-moves" neighborhood allows the descent algorithm to explore more intensively and effectively its search space and obtain better solutions than with $\mathcal{N}_{shift}$ and $\mathcal{N}_{swap}$.

Given this observation, we retain $\mathcal{N}_{bl\_moves}$ for integration in our ILS algorithm.

5.4. Iterated local search parametrization

In our ILS algorithm, three parameters (see Sections 4.4.3, 4.4.4 and 4.5) need to be tuned:

- α, the portion of neighborhood considered at each iteration,
- β, used to compute the number of idle iterations elapsed before the interruption of the local search phase,
- η, the number of accepted moves during the perturbation.

The best values for these parameters are of course interdependent. For this reason, we use a 2-level full factorial experiment in order to determine the significant parameters affecting the response of ILS (Fig. 5). We refer the reader to the book of Montgomery (2004) for a survey on the design of experiments. The levels of the parameters are indicated in Table 2.

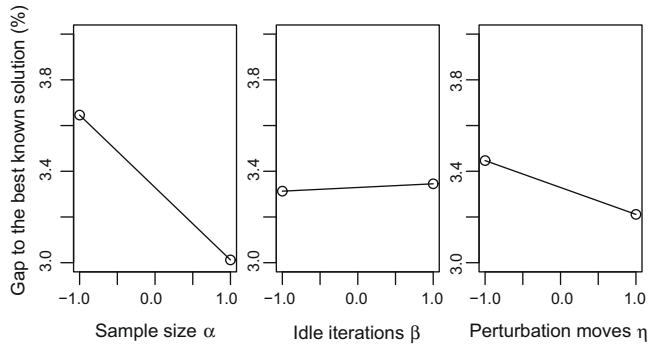

Fig. 5. Effect of the parameters on the ILS algorithm's response.

Table 2
Parameter levels (%).

	Low level	High level
α	0	0.01
β	0.1	0.5
η	10	1000

Observing Fig. 5, α appears to be a significant factor. A student t-test rejected the null-hypothesis at a 0.05 threshold confirming the graphical impression. The performance of the ILS algorithm seems less sensitive to the values of β and η. Interactions between factors do not appear to be significant.

Now setting $\beta = 0.5$ and $\eta = 1000$, we further study the influence of α in the range $[0, 0.01]$. The results of these experiments are depicted in Fig. 6.

Based on these results, we decided to retain the triplet $(0.002, 0.5, 1000)$ for α, β and η. Other values for β and η did not yield significant differences.

5.5. Comparative results

In this section, we compare ILS with 4 state-of-the-art algorithms presented in Pepin et al. (2008) using the same set of benchmarks. The studied heuristics were the following:

- a GENCOL based column generation heuristic, prematurely halted when the solution improvement is getting slow,
- a Lagrangian relaxation,
- a large neighborhood search algorithm, also based on the GENCOL package to reconstruct a solution,
- and a Tabu search algorithm using the $\mathcal{N}_{shift}$ and $\mathcal{N}_{swap}$ neighborhoods.

A heuristic MIP solver was also addressed in Pepin et al. (2008) but is not considered here. Suffering from high computational time requirements, this heuristic even failed to solve one of the largest instances within the allocated 10 h.

To have a fair comparison, each algorithm should ideally be coded by the same expert programmer and run on the same environment. In the present case, gathering such experimental conditions is impossible.

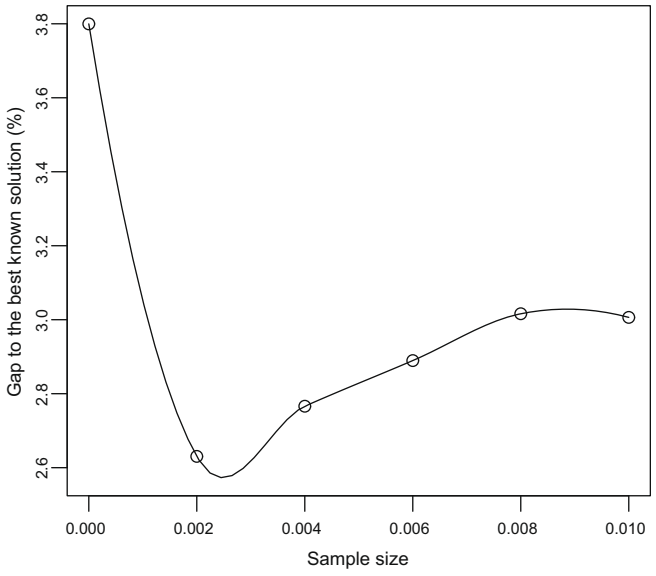

Fig. 6. Tuning of the α parameter.

Still a comparison is necessary and interesting to assess the performance of our ILS algorithm. The computing times indicated in Pepin et al. (2008) were set in function of column generation: they correspond to the minimum amount of time for this algorithm to yield good quality results. The paper indicates that it would be difficult to substantially reduce the computational time needed by column generation without significantly deteriorating solution quality. Since the Intel Xeon 2.66 GHz workstation used in Pepin et al. (2008) is slightly faster in term of CPU speed than our laptop, we decide to apply a 1.33 ratio to the computing times indicated in this paper for each instance. We use the following time limits:

- 110, 930, and 3060 s for the 4 depots 500-, 1000- and 1500-trips instances.
- 165, 1200, and 4260 s for the 8 depots 500-, 1000- and 1500-trips instances.

Table 3 presents the comparative results of ILS and the four mentioned algorithms, taken from Pepin et al. (2008). We also show the best known results, available at the above website. To report the results of ILS, this algorithm is run 10 times on each instance. The displayed results correspond to an average of averages over the 5 instances in each category ($m \in \{4,8\}$, $n \in \{500,1000,1500\}$).

From the table, one observes that the proposed ILS algorithm ranks second after the Column generation heuristic and surpasses the other heuristics. In particular, on 5 out of 6 problem categories, ILS obtains better solutions than LNS which is the best performing metaheuristic implemented so far (and which is based on the GENCOL package). Moreover, it clearly dominates TS and Lagrangian heuristic. Given its simplicity, the implemented ILS algorithm can be considered as a credible alternative to Column generation based heuristics.

5.6. Detailed results

While Table 3 presents a synthetic view of the results obtained by the different algorithms, we expose in Table 4 some statistics computed over the set of ILS runs.

Table 3
Comparison of the ILS algorithm with four best performing heuristic algorithms with respect to the best known solutions.

Depots	4			8		
Trips	500	1000	1500	500	1000	1500
Best known	1278050.80	2478517.20	3602756.40	1285485.80	2495899.80	3625731.80
Column generation	1278107.60	2478739.00	3603044.00	1285575.40	2496005.60	3625731.80
ILS	1279113.70	2479968.80	3605525.42	1286904.80	2498216.02	3629198.40
LNS	1279275.70	2480065.30	3605551.30	1286820.90	2498458.90	3629288.80
Lagrangian heuristic	1279816.90	2483082.40	3610871.40	1287924.10	2501648.90	3637341.80
Tabu search	1284342.80	2488500.30	3618857.80	1293769.60	2514100.80	3650643.00

Table 4
Detailed results of the ILS runs.

	Distance to the best known result (%)					Vehicles	Absolute costs		Opt
	Avg	SD	Med	Min	Max		Avg	Best known	
m4n500s0	1.36	0.36	1.27	1.06	2.27	123	1289917.20	1,289,114	*
m4n500s1	1.53	0.38	1.50	1.02	2.42	118	1242560.30	1,241,618	*
m4n500s2	2.67	0.27	2.68	2.18	3.13	123	1285245.10	1,283,811	*
m4n500s3	2.14	0.30	2.19	1.41	2.50	120	1,259,889	1,258,634	*
m4n500s4	1.54	0.38	1.39	1.22	2.54	126	1317956.90	1,317,077	*
m4n1000s0	2.01	0.21	2.02	1.56	2.27	241	2518377.30	2,516,247	
m4n1000s1	1.31	0.15	1.27	1.11	1.54	229	2,415,008	2,413,393	
m4n1000s2	0.91	0.14	0.88	0.69	1.11	233	2454022.40	2,452,905	
m4n1000s3	1.05	0.14	1.07	0.81	1.25	237	2492083.40	2,490,812	
m4n1000s4	0.81	0.14	0.75	0.67	1.06	238	2520352.90	2,519,229	
m4n1500s0	1.93	0.22	1.87	1.68	2.21	368	3833821.10	3,830,912	
m4n1500s1	0.93	0.12	0.93	0.80	1.10	338	3560843.60	3,559,176	
m4n1500s2	1.40	0.19	1.41	1.10	1.70	350	3651846.70	3,649,757	
m4n1500s3	2.38	0.18	2.42	2.13	2.66	326	3410310.50	3,406,815	
m4n1500s4	2.69	0.29	2.64	2.26	3.16	343	3570805.20	3,567,122	
m8n500s0	2.49	0.45	2.36	1.94	3.08	124	1293718.50	1,292,411	
m8n500s1	3.02	0.41	3.16	2.33	3.58	123	1278336.50	1,276,919	*
m8n500s2	3.11	0.43	3.15	2.62	3.85	126	1305625.90	1,304,251	*
m8n500s3	3.63	0.62	3.61	2.72	4.50	123	1279573.90	1,277,838	
m8n500s4	2.74	0.49	2.70	2.18	3.74	123	1277269.20	1,276,010	*
m8n1000s0	1.77	0.22	1.78	1.46	2.18	232	2,423,915	2,422,112	
m8n1000s1	3.13	0.17	3.14	2.77	3.33	244	2526928.80	2,524,293	
m8n1000s2	3.06	0.26	3.03	2.65	3.64	247	2558958.10	2,556,313	
m8n1000s3	1.71	0.28	1.64	1.48	2.39	237	2480250.40	2,478,393	
m8n1000s4	2.68	0.22	2.64	2.38	3.12	240	2501027.80	2,498,388	
m8n1500s0	3.08	0.43	2.99	2.66	3.98	337	3,504,165	3,500,160	
m8n1500s1	1.28	0.21	1.25	0.93	1.70	366	3,804,471.40	3,802,650	
m8n1500s2	3.41	0.36	3.43	2.85	3.94	349	3,609,013	3,605,094	
m8n1500s3	2.72	0.27	2.71	2.30	3.07	338	3519494.80	3,515,802	
m8n1500s4	3.71	0.41	3.77	3	4.41	360	3708847.80	3,704,953	

Table 5
Results of ILS with varying computational times – gap to the best solutions (%).

Depots	4			8		
Trips	500	1000	1500	500	1000	1500
Short runs	2.14	1.46	2.06	3.49	2.93	3.30
Normal runs	1.85	1.22	1.86	3.00	2.47	2.84
Long runs	1.23	1.03	1.52	1.98	1.89	2.21

The first column of Table 4 indicates the instance name. The next five columns are related to the distance to the best known solutions. They show, respectively the average values over the 10 runs (Avg), the standard deviation (Sd), the median (Med), the minimum (Min) and the maximum (Max) values. The column entitled "Vehicles" informs about the number of vehicles. The next two columns contain absolute costs, namely the average costs obtained by ILS (Avg) and the best known cost (Best known). The last column (Opt) indicates whether the instance has been solved to optimality by CPLEX or not.

It can be observed that as other heuristics, ILS easily attains the minimum number of vehicles (main objective). In addition, its results over different runs remain quite stable for most of the instances. If the best solutions ("Min" column) are considered, one observes that for 10 instances over 30, the gap to the best known one is less than 1.2%.

5.7. Variations on computational time

In this section, we are interested in the behavior of the ILS heuristic when shorter or longer computational time is provided. For this purpose, we halved the initial cutoff time allocated to ILS in a first experiment and multiplied it by five in a second experiment. The results are gathered in Table 5 (averaged over 5 runs).

This table first shows that ILS still furnishes good quality solutions within short computational times. Compared to the reference runs, we observe a loss always inferior to 0.5%. This is an interesting feature in practical situations where the algorithm can be halted at any moment. This also constitutes an important advantage with respect to the Column Generation based heuristics whose solution quality may significantly deteriorate if the allowed computational time is reduced (Pepin et al., 2008).

On the other hand, this table also confirms that ILS benefits from more computational time. Indeed, long runs lead always to solutions of better quality. The improvement is even important for most problem categories. Notice that it may be possible to accelerate the running speed of the ILS algorithm by code optimization techniques.

6. Conclusions

The multiple depot vehicle scheduling problem is of paramount importance in the operational planning process of public transport systems. Although various solution approaches have been proposed in the last three decades, the first metaheuristics to deal with the MDVSP appeared only very recently.

This article introduced an iterated local search algorithm for the MDVSP. This algorithm integrates some key features that have a great impact on the heuristic search:

- an effective auction algorithm that allows the fast construction of good quality initial solutions,
- a powerful "block-moves" neighborhood schema which maintains the good properties of the configuration and repairs conflicts as soon as they arise,
- a neighborhood sampling technique that provides a good trade-off between the computational time and the quality of the selected neighbors,
- two efficient perturbations mechanisms sequentially applied.

We have assessed the practical effectiveness of ILS by extensive experimentations using a set of 30 benchmark instances from the literature. Computational results have showed that ILS is able to furnish high quality solutions in very short computational time. Indeed, even if the column generation heuristic algorithm remains the *best* solution approach, ILS dominates the two best existing metaheuristic algorithms (Tabu search and column generation-based large neighborhood search). Furthermore, we have observed that still better solutions can be obtained when the cutoff time is increased. The ILS algorithm is also robust since the quality of its solutions varies little across different runs.

We have also carried out an analysis of the "block-moves" neighborhood and the existing neighborhood structures. Based on this analysis, we have put forward the advantage of the "block-moves" neighborhood and showed why it provides the search procedure with more search capacity.

Acknowledgements

This work was partially supported by the French Ministry for Research and Education through a CIFRE Contract (No. 176/2004). Finally, we would like to thank Valérie Guihaire for various constructive discussions. The reviewers of the paper are greatly acknowledged for their helpful comments.

References

Bertossi, A., Carraresi, P., & Gallo, G. (1987). On some matching problems arising in vehicle scheduling models. *Networks, 17*(3), 271–281.
Bertsekas, D. P. (1991). *Linear network optimization: Algorithms and codes*. Cambridge, MA: MIT Press.
Bodin, L., Golden, A., & Ball, M. (1983). Routing and scheduling of vehicles and crews: The state of the art. *Computers & Operations Research, 10*(2), 63–211.
Bunte, S., Kliewer, N., & Suhl, L. (2006). An overview on vehicle scheduling models in public transport. In *Computer-aided scheduling of public transport*. Leeds, UK: Springer Verlag.
Carpaneto, G., Dell'Amico, M., Fischetti, M., & Toth, P. (1989). A branch and bound algorihm for the multiple depot vehicle scheduling problem. *Networks, 19*(5), 531–548.
Carraresi, P., & Gallo, G. (1984). A multi-level bottleneck assignment approach to the bus drivers' rostering problem. *European Journal of Operational Research, 16*(2), 163–173.
Desaulniers, G., & Hickman, M. (2007). Public transit. In G. Laporte & C. Barnhart (Eds.). *Handbooks in operations research and management science* (Vol. 14, pp. 69–127). Amsterdam: Elsevier Science.
Desrosiers, J., Soumis, F., & Desrochers, M. (1984). Routing with time windows by column generation. *Networks, 14*(4), 545–565.
Fischetti, M., Lodi, A., Martello, S., & Toth, P. (2001). A polyhedral approach to simplified crew and vehicle scheduling problems. *Management Science, 47*(6), 833–850.
Forbes, M., Holt, J., & Watts, A. (1994). An exact algorithm for multiple depot bus scheduling. *European Journal of Operational Research, 72*(1), 115–124.
Freling, R., Wagelmans, A., & Paixão, J. M. P. (2001). Models and algorithms for single-depot vehicle scheduling. *Transportation Science, 35*(2), 165–180.
Glover, F. (1996). Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics, 65*(1–3), 223–253.
Hadjar, A., Marcotte, O., & Soumis, F. (2006). A branch-and-cut algorithm for the multiple depot vehicle scheduling problem. *Operations Research, 54*(1), 130–149.
Kliewer, N., Mellouli, T., & Suhl, L. (2006). A timespace network based exact optimization model for multi-depot bus scheduling. *European Journal of Operational Research, 175*(3), 1616–1627.
Löbel, A. (1998). Vehicle scheduling in public transit and lagrangian pricing. *Management Science, 44*, 1637–1649.
Löbel A. Optimal vehicle scheduling in public transit, November 1997. PhD thesis, Technische Universität, Berlin, 1997.
Lourenço, H., Martin, O., & Stützle, T. (2002). *Iterated local search*. Norwell, MA: Kluwer Academic Publishers.
Montgomery, D. C. (2004). *Design and analysis of experiments* (6th ed.). New York, USA: John Wiley & Sons.

Odoni, A. R., Rousseau, J.-M., & Wilson, N. H. (1994). *Models in urban and air transportation*. *Handbooks in operations research and management science* (Vol. 6). North-Holland, Amsterdam: Elsevier Science (pp. 107–150).

Oukil, A., Ben Amor, H., Desrosiers, J., & Gueddari, H. (2007). Stabilized column generation for highly degenerate multiple-depot vehicle scheduling problem. *Computers & Operations Research, 3*(34), 817–834.

Pepin, A. -S., Desaulniers, G., Hertz, A., & Huisman, D. (2008). Comparison of five heuristics for the multiple vehicle scheduling problem. *Journal of Scheduling*, doi:10.1007/s10951-008-0072-x.

Ribeiro, C., & Soumis, F. (1994). A column generation approach to the multiple-depot vehicle scheduling problem. *Operations Research, 42*(1), 41–53.

Rousseau, J.-M., Lessard, D., & Désilets, A. (1988). *Aliages: A system for the assignment of bus routes to garages*. *Computer-aided scheduling of public transport*. Berlin: Springer Verlag (pp. 8–14).

Schwarz, B. L. (1994). A computational analysis of the auction algorithm. *European Journal of Operational Research, 74*(1), 161–169.