Simultaneous Vehicle and Driver Scheduling: a Case Study in a Limousine Rental Company

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

[a] *Perez Informatique SA*
41 avenue Jean Jaurès, 67100 Strasbourg, France
[b] *LERIA, Université d'Angers*
2 Boulevard Lavoisier, 49045 Angers Cedex 01, France

Accepted in May 2007

Abstract

In this paper, we address a driver-vehicle scheduling problem in a limousine rental company. Given a set of trips to be covered, the goal consists in finding a driver-vehicle schedule that serves the maximum workload and optimizes several economic objectives while satisfying a set of imperative constraints. In this context, we propose a simultaneous scheduling of drivers and vehicles. The problem is modeled using the notion of partial consistent assignment. The solution approach is composed of two phases: the first one is based on constraint programming techniques and leads to the construction of an initial solution, improved in a second phase by a Simulated Annealing algorithm. Significant gains on the resulting solutions are systematically obtained in terms of quality, operational costs and elaboration time, compared to the current practice in the company.

Key words: Simultaneous Vehicle and Driver Scheduling, Partial Consistent Assignment, Simulated Annealing, Constraint Handling.

1 Introduction

This paper deals with a real-world driver and vehicle scheduling application in a limousine rental company. The underlying problem can be informally described as follows. We are given daily sets of trips, drivers and vehicles,

* Corresponding author. Tel.: +33 388449621; Fax:+33 388449601
Email addresses: blaurent@perinfo.com (Benoît Laurent), hao@info.univ-angers.fr (Jin-Kao Hao).

with the goal of scheduling resources in order to cover as many as possible of the trip demands. The quality of service being a crucial issue, a schedule must comply with a set of imperative constraints, while optimizing some economic objectives.

In the company organization, the design of the operational timetable is a two-phase short-term process. Every evening, according to the already booked trips (on average 70% of the total), a schedule is determined for the day after. This problem is *highly combinatorial* since instances can involve more than 200 trips, 105 drivers and 135 vehicles per day. With respect to the given rules and available resources, the problem also frequently proves to be *over-constrained*. A real-time management is then achieved throughout the day, taking into account any occurring change. Much uncertainty surrounds the trips: they can be booked, canceled or modified at the last minute according to the customers' wishes. Furthermore, the company provides services in the Paris area where congestion and delays are the daily lot. Hence, our problem takes place in a dynamic environment.

The need for a computerized solution approach can be justified for several reasons, made critical by the short-term aspect. First, there is always focus on cost reduction in the competitive market of transport. Second, the increasing workload combined with complex labor rules makes the planning process difficult to realize manually. The last, but not least, issue concerns the quality of service provided to customers.

Transportation by limousines belongs to the class of paratransit services like taxis, shuttles or demand-responsive systems (see (13)). Contrary to transit services, routes and schedules are not planned in advance. One typical example of paratransit applications is the dial-a-ride problem, denoted DARP (see (4)). The DARP and the transportation by limousines have in common a demand-responsive routing scheme. They both provide a door-to-door service. Nonetheless, the notion of ride-sharing, at the core of the DARP, is strictly excluded with limousines. The management of fleets of taxis seems to be a closer application. In (11), a taxi-dispatching problem is modeled as a Pickup and Delivery Problem with Time Windows and tackled by a steepest descent algorithm. However, the model focuses on the vehicle resource management, which is not sufficient for our situation.

In transport scheduling, the fundamental issue is the efficient use of both types of resources, crews and vehicles. This area has been a subject of continuous research since the 1950's, as attested by the abundant literature. An overview of transport scheduling can be found in (20) or (3). The complexity of the problems addressed depends on different factors. For instance, the consideration of heterogeneous types of vehicles is usually modeled as a Multiple Depot Vehicle Scheduling Problem, which was proven NP-hard by (2). The Crew Scheduling

Problem is also known to be NP-hard. We refer to (21) and (5) for a presentation of this problem. On top of that, a recent trend in transportation by bus consists in simultaneously scheduling vehicles and crews (see (6; 8; 22; 10; 7) or (12)). In our case, the fact that many drivers keep their vehicle during their entire duty makes this simultaneous approach relevant. It discloses additional degrees of freedom but coincidentally increases the complexity of the problem.

In this work, we develop a solution approach for simultaneously scheduling drivers and vehicles dedicated to the transport by limousines. To the best of our knowledge, this is the first study for this application field. More precisely, we first state a new flexible model based on the maximal partial assignment of vehicle and crew to trips. This constraint based model has the main advantage of being intuitive and adequate to deal with the numerous constraints encountered in this application. It offers much flexibility in terms of solution approaches since various metaheuristics can be devised. Then, we propose a sequential two-phase heuristic algorithm. We use a greedy algorithm combined with constraint programming techniques to build an initial solution. This initial solution is further improved by an optimization phase based on Simulated Annealing. Different types of complex neighborhood mechanisms are compared. Experimentation on a set of 10 instances with up to 200 trips per day shows that the approach is very flexible and effective.

The paper is organized as follows. The next section is dedicated to the description of the problem under consideration while a formal model based on partial consistent assignment is described in Section 3. We present in the fourth section our resolution technique relying on metaheuristics and advanced move structures. Section 5 reports computational results on real data sets. The paper ends up with a discussion on the salient aspects of the problem and with some considerations ensuing the installation of the system in the company.

In the rest of the paper, we assume the reader to be familiar with the terminology of transport. Otherwise, suitable definitions can be found in (8) or (10).

2 Problem Description

This section describes the context and the basic elements for the general understanding of the crew and vehicle scheduling process in the company. To introduce the main constraints and objectives, we state:

- a set $\mathcal{T}$ of trips, each being defined by a time and a location for the departure and the destination, a number of passengers, a set of required driver skills and a set of vehicle features,

- a set $\mathcal{D}$ of drivers, each being characterized by a daily allowed time spread, a set of skills, etc.,
- a set $\mathcal{V}$ of vehicles, each being characterized by its capacity and its features.

The problem is to find a daily assignment of driver-vehicle pairs to the trips that satisfies a set of constraints while optimizing a set of objectives.

Notice that contrary to many studies, drivers are very different from each other because of their skills.

2.1 Working Area

The rental company covers the Paris city and its suburbs, representing a surface of approximately 12 000 km^2. In accordance with the planners, this wide area has been partitioned in 26 zones. Additionally, major traffic generators, such as large hotels or airports have been precisely identified. More than 95% of the places involved in the problem match these specific locations. If it is not the case, the place is approximated to the zone center it belongs to.

Travel times between all the identified locations have been pre-computed and stored in the database. In order to avoid a null value within a zone, a threshold value has been set. Because travel times fluctuate a lot in big cities, we have introduced possible increases according to the type of day (working/holiday period) and the time range within the day.

The rental company owns a single depot where all drivers and vehicles are based. We will assume that changeovers, i.e. the change of vehicle for a driver, take place only there.

2.2 Planning Process

The planning process in the company is divided in two phases that can be qualified of "static" and "dynamic" respectively.

Every evening for the day after, a timetable is determined according to the already booked trips and available resources. There is much uncertainty upon the end time and location of the trips. These data are dependant on customers' wishes and traffic conditions. Some estimations have been computed taking into account the historical data, by type of delivered service or even by customer if already registered. A worst-case proposal is then displayed to the human operator who can modify it according to any additional information at his/her disposal. This pessimistic policy results in trips often ending slightly

before expected and allows improving changes in the schedule afterwards.

The second phase corresponds to a real-time management by the planners who take into account any events that lead to modifications in the schedule:

- an added, canceled or modified trip,
- an update of travel time (due to traffic jam for instance),
- a breakdown or any other unexpected event.

The number of dynamic requests relative to the total number of requests provides a basic measure of the degree of dynamism of our system. This ratio has been employed in the context of dynamic vehicle routing problem (see (17)). It can be extended to our situation as well. In (15), the author discerned three categories of routing systems: weakly, moderately and strongly dynamic. The differences principally hold in the tradeoff between the minimization of response time and the minimization of the costs. The average 30% value of dynamism (see Subsection 5.1) suggests a moderately dynamic system. Indeed, one can notice a balance between these objectives. For quality purpose, customers should not have to wait but it is not a life or death issue.

2.3 Constraints

To establish the driver and vehicle schedule in accordance with quality requirements and labor rules, a number of hard constraints must be satisfied. Let us consider any trip $t \in \mathcal{T}$, a vehicle $v \in \mathcal{V}$ and a driver $d \in \mathcal{D}$ assigned to t.

Capacity constraints There must be enough seats in v to accommodate all the passengers.

Category constraints The category of v corresponds to the one required. Upgrades are possibly allowed as long as they do not exceed a certain limit. A hierarchy of 7 categories is considered here.

Features constraints The vehicle v must have the equipment, e.g. a dvd player, asked for by the customer.

Skills constraints d must have all the skills needed by t, for instance a particular spoken language.

Maximum spreadover constraints The duration between the pick-up time of the first trip and the drop-down time of the last one must be less than or equal to the maximum spread time allowed for d.

Feasible sequences constraints The drivers and the vehicles must have enough time between successive trips to move from one to the other.

Pairing constraints Beforehand, human planners specify some pairs, i.e. associations between drivers and vehicles. A driver bound to a vehicle within a pair cannot work on another vehicle.

2.4 Objectives

2.4.1 Main objective

It is imperative to meet customers' trip demands. Nevertheless, the lack of resources may prevent to satisfy them all. Consequently, the first goal is to find a schedule covering as many as possible of the trip demands. From the point of view of the rental company, it is preferable to maximize the sum of the durations of the assigned trips since long trips are more profitable than short ones. In addition, the trips starting in an imminent way must be favoured. Unassigned trips are manually handled afterwards. Human planners can call additional drivers or forward the services to a subcontractor.

2.4.2 Secondary objectives: running costs

For evident economic reasons, it is desirable to reduce the number of working drivers and used vehicles. In order to further reduce costs, it is also useful to minimize the number of upgrades, the time drivers spend waiting and driving between trips. Notice that drivers do not return to the depot between trips unless they are obliged to do so for a change of vehicle. They rather stand in front of some hotel where an attendance must be guaranteed.

In the previous discussion, we have presented in a very general way the concepts that underlie our application. The main goal consists in finding an assignment of "driver-vehicle" pair for each trip which satisfies the stated constraints, while minimizing the main and secondary objectives.

In the next section, we show a formal description of the problem.

3 Problem Formulation

In this part, we present a mathematical formulation based on a partial constraint satisfaction problem, which itself is a well-known model for its flexibility and its generality for expressing complex constraints. Notice that this model is

not intended for an exact resolution algorithm. Indeed, as we see later in this section, the complexity of our application (rich constraints, multi-objectives...) and the practical requirements (relatively short computing times...) prevent us from applying any time-consuming method (Branch and Bound...). This point is further discussed in Subsection 6.1.

3.1 Partial Consistent Assignment

The model of partial consistent assignment is described in (1) as a way to cope with over-constrained problems. The idea behind this model is to assign as many variables as possible without getting inconsistency, i.e. constraint violation. Because we seek a solution obtainable within fixed resource bounds (pre-determined number of vehicles, etc.) and within a short delay, such an approach is relevant for our application. Notice that in our case, we take into account other optimization objectives than the cardinality of an assignment. The following definitions formalize the notion of partial consistent assignment.

Let us consider a triplet $(X, Dom, Const)$ where X is a finite set of variables, Dom a set of associated domains Dom_i specifying the possible values for each variable and $Const$ a set of constraints restricting the values of variables.

Definition 1 *An assignment σ is a set of pairs (x_i, val_i) such that $x_i \in X$, $val_i \in Dom_i$ and each x_i appears at most once in σ. The assignment is consistent if it satisfies the set of given constraints over the instantiated variables.*

Definition 2 *An assignment σ is partial if there exists at least one variable of X that does not appear in σ, otherwise, the assignment is complete.*

Definition 3 *A maximal consistent assignment is a consistent assignment of the largest cardinality.*

A maximal consistent assignment may be partial or complete. If the given problem is over-constrained, a maximal consistent assignment will be partial. Before defining the triplet $(X, Dom, Const)$ for our application, some useful notations are defined in the next section.

3.2 Notations

Recall that $\mathcal{T}$, $\mathcal{D}$ and $\mathcal{V}$ represent respectively the sets of trips, drivers and vehicles. Let T, D and V be their associated cardinals, i.e. $\mathrm{T} = |\mathcal{T}|$, $\mathrm{D} = |\mathcal{D}|$ and $\mathrm{V} = |\mathcal{V}|$. Given $t \in \mathcal{T}$, $d \in \mathcal{D}$, $v \in \mathcal{V}$, we state the notations figuring in Tables 1 and 2.

Table 1
Notations

$pas(t)$	number of passengers for t
$capa(v)$	capacity of v
$Cat(t)$	category demanded by t ($Cat(t) \in \mathbb{N}$)
$cat(v)$	category of v ($cat(v) \in \mathbb{N}$)
$st(t), et(t)$	respectively start and end time of t
$S_{\max}(d)$	maximum spread time allowed for d
$U_{\max}$	maximum upgrade allowed (possibly set to ∞)
$dh(t_k, t_l)$	deadhead time between trips t_k and t_l
$wt(t_k, t_l)$	waiting time between trips t_k and t_l

A total order $\prec$ on the set of trips $\mathcal{T}$ is defined by:

$$\forall (t_k, t_l) \in \mathcal{T}^2, t_k \prec t_l \Leftrightarrow \begin{cases} st(t_k) < st(t_l) \\ \vee \\ (st(t_k) = st(t_l) \ \wedge \ k < l) \end{cases}$$

We consider a set of skills $\mathcal{S}$ and a set of features $\mathcal{F}$ to define the binary relations listed in Table 2.

Table 2
Binary relations

$sk(d, s)$	driver d owns skill s
$Sk(t, s)$	trip t requires skill s
$ft(v, f)$	vehicle v owns feature f
$Ft(t, f)$	trip t demands feature f
$p(d, v)$	driver d and vehicle v are tied within a pair
$compat(t_k, t_l)$	trips t_k and t_l can be realized by the same resources

In addition, we introduce the following notations to handle "driver-vehicle to trip" assignments:

- $wd(d)$, (resp. $vu(v)$) $\Leftrightarrow$ d (resp. v) is assigned to at least one trip,
- $Seq(d)$ is the set of pairs (t_k, t_l) that $d \in \mathcal{D}$ handles consecutively.

Finally, we define $compat'(t_k, t_l), dh'(t_k, t_l), wt'(t_k, t_l)$ and $Seq'(d)$ that are similar to $compat(t_k, t_l), dh(t_k, t_l), wt(t_k, t_l)$ and $Seq(d)$ respectively except that they take into account a stop at the depot between trips.

3.3 Decision Variables and Domains

In our application, we may identify the set of trips $\mathcal{T}$ as the set of our decision variables.

Each $t_k \in \mathcal{T}$ is possibly assigned a value in its domain $\mathcal{I}_k$, which corresponds to driver-vehicle pairs. Initially, all domains $\mathcal{I}_k$ are equal to $\mathcal{I} = \mathcal{D} \times \mathcal{V}$.

The set of constraints is exposed in Subsection 3.4, the pursued objectives in Subsection 3.5.

3.4 Constraints

We provide hereafter a mathematical definition of the types of constraints informally described in Subsection 2.3.

Capacity constraints

$$\forall\, t \in \mathcal{T}, \forall\, v \in \mathcal{V},\ t = (., v), \quad CAPA(t, v) \Leftrightarrow pas(t) \leq capa(v)$$

Category constraints

$$\forall\, t \in \mathcal{T}, \forall\, v \in \mathcal{V},\ t = (., v), \quad CATEGORY(t, v) \Leftrightarrow 0 \leq cat(v) - Cat(t) \leq U_{\max}$$

Features constraints

$$\forall\, t \in \mathcal{T}, \forall\, v \in \mathcal{V},\ t = (., v),\ \forall\, f \in \mathcal{F},\ FEATURE(t, v, f) \Leftrightarrow \neg Ft(t, f) \vee ft(v, f)$$

Skills constraints

$$\forall\, t \in \mathcal{T}, \forall\, d \in \mathcal{D},\ t = (d, .),\ \forall\, s \in \mathcal{S}, \quad SKILLS(t, d, s) \Leftrightarrow \neg Sk(t, s) \vee sk(d, s)$$

Maximum spread time constraints

$$\forall\, d \in \mathcal{D}, \forall\, (t_k, t_l) \in \mathcal{T}^2,\ t_k = (d, .),\ t_l = (d, .),\ t_k \prec t_l,$$

$$MAX\_SPREAD(t_k, t_l, d) \Leftrightarrow (et(t_l) - st(t_k)) \leq S_{\max}(d)$$

Feasible sequences constraints

$\forall\, d \in \mathcal{D}, \forall\, (t_k, t_l) \in \mathcal{T}^2,\ t_k = (d, .),\ t_l = (d, .),\ t_k \prec t_l,$

$$FEASIBLE\_D(t_k, t_l, d) \Leftrightarrow \begin{cases} compat(t_k, t_l)\ \wedge\ (t_k, t_l) \in Seq(d) \\ \vee \\ compat'(t_k, t_l) \end{cases}$$

Similar constraints exist for the vehicles.

Pairing constraints

$$\forall\, t \in \mathcal{T}, \forall\, (d, v) \in \mathcal{I},\ t = (d, v), \quad PAIRING(t, d, v) \Leftrightarrow \forall v' \in \mathcal{V}\ \neg p(d, v') \vee v = v'$$

3.5 Objectives

The evaluation function is the weighted sum of different criteria described hereafter. The way in which these objectives are aggregated is explained in Subsection 4.3.3.

Main objective

Minimizing the total duration of the uncovered trips:

$$Min \sum_{t \notin \sigma} (et(t) - st(t)), \quad \sigma \text{ being an assignment}$$

Viewed another way, we aim at finding a maximal weighted consistent assignment, each variable being weighted by its duration.

Secondary objectives

Minimizing the number of working drivers and vehicles in use:

$$Min(\sum_{d \in \mathcal{D}} wd(d) + \sum_{v \in \mathcal{V}} vu(v))$$

Minimizing upgrades:

$$Min \sum_{t \in \sigma\, \wedge\, v \in \mathcal{V}\, \wedge\, t = (.,v)} (cat(v) - Cat(t))$$

Minimizing deadheads:

$$Min \sum_{d \in \mathcal{D}} \left(\sum_{(t_k,t_l) \in Seq(d)} dh(t_k,t_l) + \sum_{(t_k,t_l) \in Seq'(d)} dh'(t_k,t_l) \right)$$

Minimizing the total waiting time:

$$Min \sum_{d \in \mathcal{D}} \left(\sum_{(t_k,t_l) \in Seq(d)} wt(t_k,t_l) + \sum_{(t_k,t_l) \in Seq'(d)} wt'(t_k,t_l) \right)$$

4 Solution Approach

Our solution method consists in a two-phase algorithm. The first step relies on constraint programming techniques and leads to the construction of an initial solution. This solution is then improved by a Simulated Annealing (SA) algorithm. A particular attention has been payed to the definition and the assessment of powerful neighborhood exploration mechanisms.

4.1 Constraint based pre-processing

The number of potential assignments is exponential subject to the cardinality of the set of decision variables and to the cardinality of the domains. To reduce this number, we borrow filtering techniques from constraint programming (19). The node consistency property allows to remove from domains all values that are inconsistent with at least one constraint. Let us consider, for example, the PAIRING constraints:

$$\forall\, t_k \in \mathcal{T}, \forall\, (d,v) \in \mathcal{I},\ (\exists\ v'\ \in \mathcal{V},\ v' \neq v,\ p(d,v'))\ \Rightarrow (d,v) \notin \mathcal{I}_k$$

If a driver is bound to a vehicle within a pair, the values corresponding to this driver associated with any other vehicle will be removed from the domain of all variables.

4.2 Initial Schedule

Two ways of constructing the initial solution exist.

The day before, the initial solution is built from scratch by a constructive greedy heuristic sketched hereafter. The strategy employed recalls the Best

Fit Decreasing Strategy introduced in (18) to tackle the Bin Packing Problem. The variables (the trips) are sorted in decreasing order of duration, subsequently labelled one by one by a driver-vehicle pair that can handle it. Already employed resources are of course privileged. After each assignment, a forward checking procedure is applied to prevent future conflicts. It performs arc consistency between the currently instantiated variable and the not-yet instantiated ones. Considering the MAX_SPREAD constraints for example, once a driver-vehicle (d_k, v_k) has been assigned to a variable t_k, all pairs including d_k are removed from the domain of the variables bound to t_k within constraints of type MAX_SPREAD.

$$\forall\, t_j \in \mathcal{T}, t_j \prec t_k,\ (et(t_k) - st(t_j) \geq S_{max}(d_k)) \ \Rightarrow\ \forall v \in \mathcal{V},\ (d_k, v) \notin \mathcal{I}_j \,^{1}$$

The construction procedure stops either when a complete assignment is obtained or when no additional assignment can be made without violating constraints.

Algorithm 1: Initial schedule construction (day before)

Data : $\mathcal{T}$ the set of variables, $\mathcal{I}_k$ the sets of associated domains
Result: Initial configuration
begin
  /* Sort variables by decreasing order of duration */
  sort_decreasing($\mathcal{T}$)
  /* Unlabelled variables */
  $Unlab \leftarrow \mathcal{T}$
  /* Iterate through unlabelled variables */
  for $t_k \in Unlab$ **do**
    if $\mathcal{I}_k \neq \emptyset$ **then**
      /* Provide best pair driver-vehicle */
      $t_k \leftarrow$ best_value(I_k)
      $Unlab \leftarrow Unlab - t_k$
      /* Apply forward checking procedure on unlabelled variables */
      forward_checking($t_k, Unlab$)
    end
  end
end

In the real-time management, the current schedule is loaded as initial solution with possibly slight alterations. Indeed, some variables may be unassigned to avoid broken constraints. Violations can happen for instance when a delay in a trip causes infeasible sequences.

[1] We imposed $t_j \prec t_k$ to simplify the formula but the symmetric case is similar.

The initial solution is then improved by a Simulated Annealing algorithm (see (14)) which main components are exposed in the next section.

4.3 Simulated Annealing

4.3.1 General algorithm

The main ingredients of SA are: an initial temperature, a rule for accepting a damaging move, another for decrementing the temperature, a maximum number of neighboring solutions that can be generated at each temperature and a stop criterion. The initial temperature t_0 is set according to the presumed quality of the initial solution. Here, the acceptance of damaging moves is controlled by the Metropolis rule and the cooling schedule by a geometrical scheme ($t_k = \alpha t_{k-1}$). At each temperature threshold, two parameters can trigger a decrementation:

- the number of accepted states,
- the number of visited states.

Finally, different criteria may stop the algorithm: a maximum duration, a predefined number of thresholds without improvement or a temperature close to zero.

4.3.2 Search space

A configuration σ is a consistent assignment, not necessarily complete, of "driver-vehicle" pairs in $\mathcal{I}$ to trips in $\mathcal{T}$ (see Figure 1).

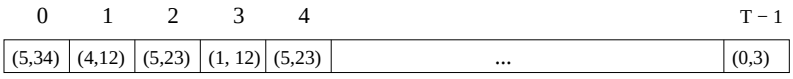

Fig. 1. Configuration representation

The search space Ω is then defined as the set of all such assignments. Hence, Ω may contain partial assignments but no infeasible solutions.

Notice that our representation remedies in essence, i.e. without any additional constraint, some drawbacks raised by set partitioning or set covering models largely used for the Crew Scheduling Problem. The former is rather hard to solve and is often relaxed in the set covering formulation. However, the latter can lead to over-covered trips (see (12) for instance).

4.3.3 Evaluation function

In order to guide the algorithm to visit the search space, one needs a function to evaluate the configurations. The quality of a configuration σ is estimated through a weighted aggregation of the different objectives.

$$\forall \sigma \in \Omega, \quad eval(\sigma) = \sum_{o \in \mathcal{O}} w_o \times f_o(\sigma)$$

with:

- $\mathcal{O}$ the set of objectives,
- $w_o > 0$ the weight associated to the o^{th} objective function,
- f_o the value of o^{th} objective function on σ.

The difficulty here lies in the combination of diverse objectives, different in nature and scale. Normalizing them on the interval [0, 1] would have required the knowledge of the lower and upper bounds, which are hard to estimate. Instead, we record the results per criterion obtained on the previous runs relative to the number of trips. The mean m_o on each criterion is sent as a parameter for a new run. The weight associated to the o^{th} objective function is computed in this way:

$$w_o = \alpha_o \times \frac{1}{m_o \times \mathrm{T}} \qquad \alpha_o \text{ being specified by the user}$$

We thus obtain a value $\frac{1}{m_o \times \mathrm{T}} \times f_o(\sigma)$ close to 1.

Figure 2 shows the sequence m_o over 60 runs on different instances for the number of drivers criterion. The curve stabilizes over time. Indeed, the system benefits from past experiences and has the potential to estimate the number of required drivers.

The vector of weights α may vary according to each user's appreciation of the relative importance of the criteria. In Table 3, a combination empirically determined for the tests is presented. The value for the unassigned work reflects the importance of this objective.

Table 3
α weights

unassigned work	drivers	vehicles	upgrades	deadheads	waiting time
100	1	1	0.1	0.01	0.005

4.3.4 Neighborhood operators

It is well known that the neighborhood is one of the most important component of any SA algorithm. For this reason, we have defined and experimented

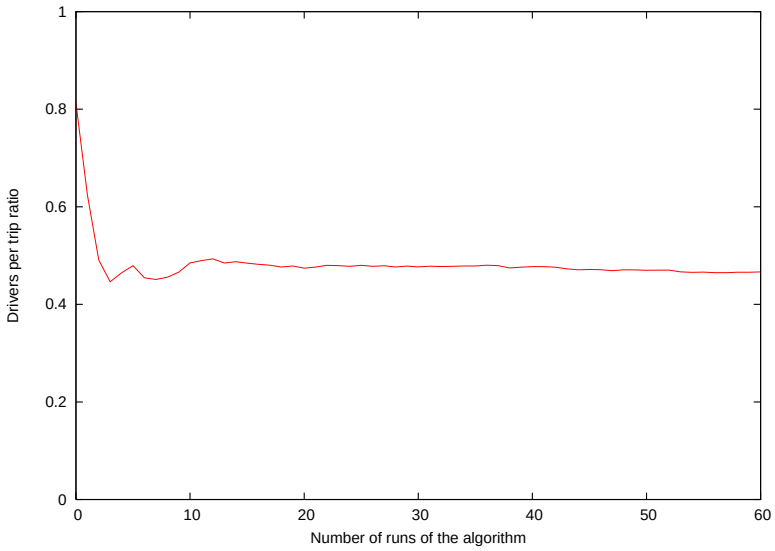

Fig. 2. Average number of drivers per trip on different runs

different neighborhood operators ranging from simple to complex compound moves.

1_change It affects one variable. The step from one configuration to another is achieved in this way:

> (1) pick a variable (already instantiated or not) at random,
> (2) pick a new value within its domain,
> (3) if assigning the value to the variable does not create any conflict, the assignment becomes effective, otherwise go to step 1.

Depending on the case, *1_change* might correspond in reality to:

- a change of driver,
- a change of vehicle,
- a change affecting both resources.

Swap This neighborhood mechanism is quite intuitive:

> (1) pick two variables, one at least being assigned,
> (2) exchange their values if consistency is preserved, otherwise go to step 1.

This neighborhood mechanism can be seen as two particular simultaneous applications of *1_change*.

1_change_&_re-assigns This neighborhood mechanism derives from *1_change*. The sketch of a move is outlined below:

(1) pick a variable (already instantiated or not) at random,
(2) pick a new value within its domain,
(3) if assigning the value to the variable does not create any conflict, the assignment becomes effective and the ancient value of the variable is stored, otherwise go to step 1.
(4) loop through unassigned variables and try to assign them the value previously recorded.

The strength of this neighborhood mechanism lies in the fourth step. Remember that in the initial schedule, no additional variable can be assigned without constraint violation. Transposed to the reality, it means that the resources are all unavailable or can not be used because of some constraints. Applying *1_change* may release some resources that can be assigned to another trip.

Ejection_chain The ejection chains were introduced by (9) who defined them 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 the elements of at least one must be "ejected from" their current state. We stated an ejection chain in our context as follows:

(1) pick a variable (already instantiated or not) at random,
(2) pick a new value within its domain,
(3) if assigning the value to the variable does not create any conflict, the assignment becomes effective, otherwise, a new value is assigned to all the variables in conflict if a consistent assignment is possible. As a last resort, these variables are unassigned.

This powerful neighborhood operator may generate some conflicts that it immediately repairs. However, like *1_change*, it suffers from the weakness of not maintaining the full employment of the resources.

Ejection_chain_&_re-assigns *Ejection_chain_&_re-assigns* merges the two former neighborhood operators. It overcomes their weaknesses and takes advantage of their respective strengths. It works as follows:

(1) pick a variable (already instantiated or not) at random,
(2) pick a new value within its domain and store the ancient value,
(3) if assigning the value to the variable does not create any conflict, the assignment becomes effective, otherwise, a new value is assigned to all the variables in conflict if a consistent assignment is possible. As a last resort, these variables are unassigned.
(4) loop through unassigned variables and try to assign them the value previously recorded.

5 Experimentations and Results

5.1 Data and Experimental Settings

Computational experiments were conducted on 10 instances coming from the limousine rental company and representing different workloads. Table 4 provides some characteristics of these instances, namely, the problem size, the number of scheduled trips, their average duration, the available drivers and vehicles, the number of specified "driver-vehicle" pairs, and eventually the degree of dynamism (see Subsection 2.2).

Table 4
Characteristics of the instances

	Problem size	Trips	Average duration (hh:mm)	Drivers	Vehicles	Pairs specified	Degree of dynamism (%)
08_05_05	7497^{91}	91	1:28	51	147	35	28.6
10_05_05	10720^{125}	125	1:48	80	134	63	27.2
18_05_05	12555^{153}	153	2:57	93	135	64	30.1
19_05_05	12015^{163}	163	3:05	89	135	71	26.4
23_05_05	14175^{202}	202	2:15	105	135	70	27.7
07_01_06	5194^{117}	117	2:39	49	106	42	30.7
08_01_06	4558^{108}	108	2:33	43	106	35	33.3
09_01_06	5130^{130}	130	1:57	57	90	50	30
10_01_06	5187^{135}	135	2:01	57	91	52	25.9
11_01_06	5673^{159}	159	2:23	61	93	57	39

Our search procedures were coded in C++, compiled with gcc 3.4.2 (flag -03), on a PC running Windows XP (1Go RAM, 2.8Ghz). Due to the incomplete and non-deterministic nature of the methods, 20 independent runs were carried out on each instance with different random seeds. A maximum CPU time

of 10 minutes is defined as stop condition. These rules prevail for all the experimentations.

5.2 Results of the Pre-Processing Phase

The goal here is to show the effects of the pre-processing phase, described in Subsection 4.1.

In Table 5, we indicate the problem size after the pre-processing phase, the average (avg) and the standard deviation (sd) for the number of possible drivers, vehicles and pairs per trip. The last column shows the number of empty domains.

These computations underline the importance of this pre-processing phase in the drastic reduction of the initial search space. The number of matching resources per trip is rather high on average but the standard deviation, figuring into brackets, reveals a considerable disparity between trips. The number of non-assignable trips confirms the over-constrained nature of the problem.

Table 5
Results of the pre-processing phase

	Problem size	Drivers per trip avg (sd)	Vehicles per trip avg (sd)	Pairs per trip avg (sd)	Trips N/A
08_05_05	1265^{91}	32.6 (11.6)	66.5 (36.0)	614.8 (389.4)	0
10_05_05	1138^{125}	45.0 (16.9)	58.3 (35.8)	504.6 (345.5)	1
18_05_05	1923^{153}	42.4 (20.9)	64.0 (33.7)	881.7 (709.8)	2
19_05_05	1158^{163}	43.1 (20.2)	64.8 (33.7)	558.9 (413.7)	0
23_05_05	1776^{202}	48.9 (22.6)	70.8 (32.7)	885.6 (564.1)	2
07_01_06	402^{117}	14.6 (8.4)	31.0 (14.5)	79.0 (71.2)	2
08_01_06	432^{108}	13.5 (9.4)	35.6 (21.4)	103.8 (136.8)	5
09_01_06	305^{130}	16.6 (12.1)	32.8 (16.4)	77.9 (98.3)	4
10_01_06	203^{135}	18.5 (11.7)	33.5 (16.8)	72.2 (68.3)	3
11_01_06	260^{159}	22.4 (14.5)	31.8 (14.5)	57.7 (67.1)	10

5.3 Neighborhood Comparison

In this section, we compare the respective efficiency of the five neighborhood operators described in Subsection 4.3.4 by observing their effect on the combined objective function during the search.

For these experimentations, we decided to use a simple but pure Hill Climbing algorithm. Despite its tendency to fall into local optima and its incapacity to escape from them, Hill Climbing is a neutral algorithm, requiring almost no

tuning of parameters, and thus, particularly adequate for comparing different neighborhoods. Apart from the maximum duration of 10 minutes, another stop criterion is used based on the number of iterations elapsed from the last strict improvement. Otherwise, the search might loop among a restricted set of neighbors. 200 * T iterations without improvement indicate that the algorithm stagnates on a local optimum. Remember that T is the number of decision variables.

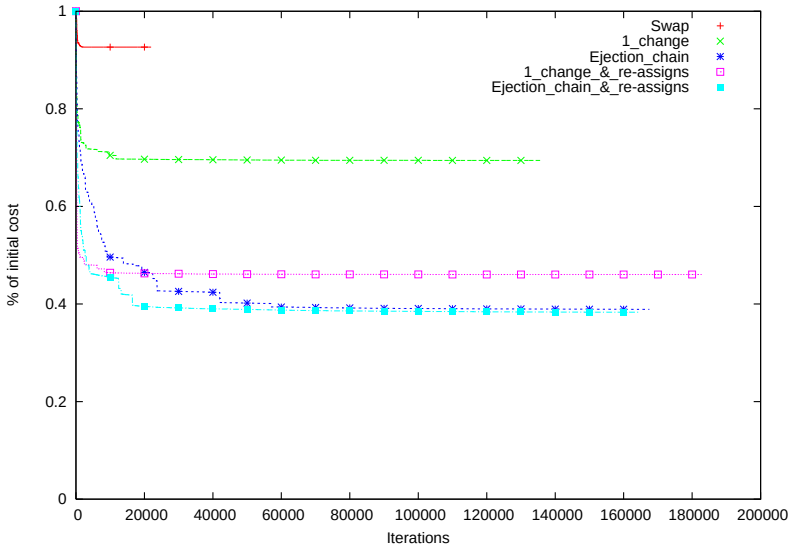

Fig. 3. Evaluation function curves by neighborhood mechanism (instance 19_05_05)

Figure 3 shows the evolution of the objective function averaged over 20 runs for each neighborhood mechanism. The values are provided as a percentage of the initial cost. The tested instance was arbitrarily chosen since all diagrams present an similar shape. The obtained ranking is always the same, variations between curves being just more or less pronounced.

¿From Figure 3, it can be observed that the *swap* mechanism behaves very poorly and is quickly trapped in a local minimum. The reason for this is that the evaluation function is dominated by the weighted number of assigned variables. The *swap* mechanism leaves this number unchanged and thus, can not improve the objective function for long. *1_change* modifies the value of a variable without creating conflicts. Its results outperform those of the *swap* neighborhood but still remain unsatisfactory. Again, the algorithm rapidly falls into a local minimum. Furthermore, the decrease is too slow to be employed in a real-time context. With its aggressive search strategy, *Ejection_chain* overcomes the first difficulty encountered by *1_change*. The curve of *1_change_&_re-assigns* delineates a rapid decrease since this neighborhood constantly seeks a maximal consistent assignment. Eventually, *Ejection_chain_&_re-assigns* combines the respective strengths of the two previous neighborhoods and provides the best results.

5.4 Comparison between Manual Scheduling and Simulated Annealing

In the previous paragraph, *Ejection_chain_&_re-assigns* emerged as the most powerful neighborhood mechanism. It was therefore embedded in a Simulated Annealing algorithm to be compared to the manually obtained results.

5.4.1 Parameter tuning

The tuning process of the SA algorithm is a delicate issue. For each parameter evoked in Subsection 4.3.1, we empirically determined an interval of reasonable size. Different combinations were tested on a sample of 20 runs resulting in the following set of parameters:

- an initial temperature of 10^{-5},
- a cooling rate of 0.95,
- a number of accepted states equal to T,
- a number of attempted configurations on each threshold of 100 * T.

Apart from the maximum duration of 10 minutes, the algorithm stops either if the current temperature is low enough, or when it ceases to make progress. Three consecutive threshold diminutions without any acceptance indicate a frozen situation.

5.4.2 Computational results

The manually generated timetables (see Table 6) constitute our references for comparison. Table 7 shows the average results obtained with our approach relying on SA, on 20 independent runs for each instance. We recall that the rules slightly differ between manual and computed schedules: in the second case, broken constraints are strictly forbidden but unassigned trips are allowed and manually handled afterwards. Hence, the column indicating the number of broken constraints in Table 6 is replaced by the one of assigned work in Table 7. The remaining columns are common to both tables and report the numbers of required drivers and vehicles, the total number of upgrades, the total durations of deadheads and idle time.

Applying strict comparison criteria between both sets of solutions is rather difficult. All the same, computerized solutions are considered profitable if they fulfill the following conditions:

- at least 80% of the workload is automatically assigned,
- the constraints are all satisfied,
- the operational costs are reduced,

• the solution is displayed within a short amount of time (10 min).

Table 6
Results of manual scheduling

Date	broken constraints	drivers	vehicles	upgrades	deadheads (hh:mm)	waiting time (hh:mm)
08_05_05	27	44	52	55	26:45	121:44
10_05_05	48	65	70	60	27:50	190:27
18_05_05	47	74	74	84	31:59	208:35
19_05_05	48	79	77	87	33:12	176:13
23_05_05	79	84	81	96	40:25	270:02
07_01_06	53	47	50	70	76:12	123:51
08_01_06	72	43	44	55	66:59	80:23
09_01_06	48	58	57	72	96:45	168:58
10_01_06	57	54	54	87	88:54	197:06
11_01_06	50	60	64	80	105:54	172:08

Table 7
Results of Simulated Annealing

Date	assigned work (%)	drivers	vehicles	upgrades	deadheads (hh:mm)	waiting time (hh:mm)
08_05_05	100	31.60	33	5.4	55:16	38:01
10_05_05	99.44	42.20	44.4	12.4	69:47	48:44
18_05_05	96.55	66.10	63.2	14.3	84:44	70:10
19_05_05	100	67.2	66.4	27.1	85:14	72:59
23_05_05	84.22	73.2	68.4	11.7	107:38	90:23
07_01_06	81.47	46.3	45.8	57.1	78:49	63:05
08_01_06	76.71	41.3	41.3	51.8	70:23	52:04
09_01_06	84:48	45.1	44.2	43.9	86:50	86:27
10_01_06	91.53	49.1	49.8	55.1	91:15	90:02
11_01_06	78.60	52.2	51.9	39.3	106:33	83:28

Figures in Table 7 indicate that the two first goals are attained. Most of the work is covered without breaking any constraint. Instances 08_01_06 and 11_01_06 must be considered separately: at the end of the pre-processing phase, the numbers of non-assignable trips detected were 5 and 10 respectively (see Table 5). In other words, it was not possible to satisfy all the customers' requirements in those instances with the available resources.

Our algorithm also yields significant gains in the operational costs. The comparison may appear tendentious as the trips are rarely all covered in computerized solutions. However, the numbers of used resources, upgrades and total waiting time are so dramatically reduced that they remain in fine far below the values of the manual timetables. Completely covered instances confirm this allegation. Notice that the constraint satisfaction has a price in terms of deadheads. Drivers are sometimes required to cover more distance to reach the start of a trip they possess the skills for. Globally speaking, the savings procured by the system are very satisfactory. Such a gap between the two sets of

solutions can be explained by the following reason: the workload for schedulers was so high that costs reduction represented only a secondary concern.

This leads to the other major contribution of our work: the time spent to elaborate a timetable is drastically decreased. Whereas people spent up to 4 hours elaborating an operational timetable, our program only takes a few minutes to produce a better quality schedule.

6 Discussions

In this section, we discuss the difficult aspects of our application that explain some of our choices. We also relate the installation of the software in the company.

6.1 Problem Analysis

This problem combines different aspects that make it difficult to solve.

First, the complexity of the constraints in this real-world problem leads us to opt for a natural modeling in terms of Partial Constraint Satisfaction Problem. The number of non-assignable trips in Table 5 underlines the over-constrained nature of the problem, or at least, highlights an inadequacy of the resources with the tasks to be accomplished.

The second difficulty is directly related to the computational time. In this study, we performed an *a posteriori* treatment that cannot render the urgency atmosphere sometimes reigning in real-life conditions (see Subsection 2.2). We thus preferred local search approaches over exact methods for their ability to produce solutions, even partial, within a short amount of time.

The results reported in Section 5 show a superiority of the computed solutions over the manually generated ones. However, the estimation of good lower bounds could give a better insight of the quality of the solutions. Since the main objective is the minimization of used resources, we first attempted to compute a lower bound on the number of vehicles, independently of the crews. The problem restricted to its vehicle aspect was modeled as a multi-commodity flow formulation (see e.g. (16)) and subsequently solved with a Branch & Bound algorithm. Unfortunately, this methodology did not furnish relevant results. For example, we obtained on the 07_01_06 instance a lower bound of 32 vehicles against an average value of 45.8 in our solutions and of 50 in the manual scheduling. Such a gap comes from the constraints binding drivers and vehicles. In the optimal solution of the vehicle problem, many trips have to be

transferred to other vehicles since the driver that they are bound to does not possess the appropriate skills. The presence of several optimization objectives makes the task of computing lower bounds even more difficult.

The last issue was out of the scope of this study but deserves to be mentioned. In the planning process of the company, duties generated are directly assigned to drivers day after day without any crew rostering phase. One could be concerned with the respect of long-term labor rules. This question can be partly taken into account through the $S_{max}(d)$ value (see 1). A driver who nearly reaches its time limit per week or per month, will work less by the end of these periods. However, this solution is not satisfactory in practice. The issue is still manually handled at the present time.

6.2 Installation

The solution approach described in this paper has been integrated into the decision support system of the company.

The software matches well the organization of the company described in Subsection 2.2. Each evening for the next day, it is launched for 10 minutes using known data to get a rough timetable. The next day, the schedule is refined whenever a new event arises. The planners usually let the system search for 1 minute, which corresponds to about 15 000 iterations depending on the machine. This duration constitutes an acceptable tradeoff between the response-time and the quality of the solution. The computerized timetable may undergo some alterations before application in practice. These manual modifications aim at integrating exceptional considerations not taken into account in the solution approach such as the preferences of a customer for a particular driver.

The decision support system has significantly lightened the labor of the human planners and allows them to greatly increase their productivity. Moreover, their job becomes less stressful and more accessible to unexperienced planners.

7 Conclusion

In this paper, we proposed a simultaneous approach for a driver and vehicle scheduling problem in an original context. To the best of our knowledge, the work reported here would be the first study in the targeted field.

The formulation as partial constraint satisfaction problem reveals to be adequate for this overconstrained problem. The non-instantiated variables disclose

the hard part of the problem. Subsequently, the user can take the appropriate decisions - calling some additional drivers, choosing the constraints to be violated - thanks to his perception of the current situation.

We applied a sequential two-phase algorithm to our model: a pre-processing phase leading to the construction of an initial solution is followed by an optimization phase using Simulated Annealing. Different neighborhood functions have been developed and assessed. The study carried out confirmed that powerful neighborhood mechanisms are essential to reach good performance in local search.

Finally, results obtained on real data sets show a significant improvement compared with the actual practice. Within a short time, the software supplies very good quality schedules in which the major part of the trips is assigned. The constraints are all satisfied whereas the operational costs, including the number of resources, the number of upgrades and the total idle time are reduced. Our approach also proves to be flexible. It unifies the treatment of the static and dynamic parts of this problem in a single framework. The decision support system based on this research is operating in the company and proves to be extremely useful.

Let us give a last comment to conclude the paper. This paper deals with a real-world driver and vehicle scheduling problem in a particular application context. Even if some aspects are specific, others are general ones. In particular, the notion of simultaneous scheduling of drivers and vehicles is quite general and relevant to many other scheduling applications. For instance, we have successfully transposed the basic constraint-based model and some solution techniques of the current work to the domain of transportation by bus in rural areas.

Acknowledgements

This research was partially supported by the French Ministry for Research and Education through a CIFRE contract (number 176/2004). The reviewers of the paper are greatly acknowledged for their comments that contributed to improve this paper. Finally, we would like to thank Frédéric Lardeux, Valérie Guihaire and Christopher Ineson for various constructive discussions.

References

[1] Barták, R. and Müller, T. and Rudová, H., 2003. A New Approach to Modeling and Solving Minimal Perturbation Problems. In: Recent

Advances in constraints, Lecture Notes in Artificial Intelligence 3010, Springer Verlag, 223-249.

[2] Bertossi, A.A. and Carraresi, P. and Gallo, G., 1987. On Some Matching Problems Arising in Vehicle Scheduling Models. Networks, 17, 271–281.

[3] Ceder, A., 2002. Urban Transit Scheduling: Framework, Review and Examples. Journal of Urban Planning and Development, 128 (4), 225–244.

[4] Cordeau, J.-F. and Laporte, G., 2003. The Dial-a-Ride Problem (DARP): Variants, Modeling Issues and Algorithms. 4OR, 1 (2), 89–101.

[5] Fores, S. and Proll, L. G. and Wren, A., 1999. An improved ILP system for driver scheduling. Computer-Aided Transit Scheduling, In: Wilson, N. H. M. (Ed.), Springer Verlag, Berlin, 43–62.

[6] Freling, R. and Boender, G. and Paixão, J. M. P., 1995. An integrated approach to vehicle and crew scheduling. 9503/A, Economie Institute, Erasmus University Rotterdam, Rotterdam.

[7] Freling, R. and Huisman, D. and Wagelmans, A. P. M, 2003. Models and Algorithms for Integration of Vehicle and Crew Scheduling. Journal of Scheduling, 6 (1), 63–85.

[8] Gaffi, A. and Nonato, M., 1999. An Integrated Approach to the Extra-Urban Crew and Vehicle Scheduling Problem. Computer-Aided Transit Scheduling, In: Wilson, N. H. M. (Ed.), Springer Verlag, Berlin, 103–128.

[9] Glover, F., 1996. Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems. Discrete Applied Mathematics, 65 (1-3), 223–253.

[10] Haase, K. and Desaulniers, G. and Desrosiers, J., 2001. Simultaneous Vehicle and Crew Scheduling in Urban Mass Transit Systems. Transportation Science, 35 (3), 286–303.

[11] Horn, Mark E. T., 2002. Fleet Scheduling and Dispatching for Demand-Responsive Passenger Services. Transportation Research part C, 10 (1), 35–63.

[12] Huisman, D., 2004. Integrated and Dynamic Vehicle and Crew Scheduling. Tinbergen Institute, Erasmus University Rotterdam.

[13] Jayakrishnan, R. and Cortés, C. E. and Lavanya, R. and Pagès, L., 2003. Simulation of Urban Transportation Networks with Multiple Vehicle Classes and Services. In: 82th Transportation Research Board Annual Meeting, Washington D.C..

[14] Kirkpatrick, S. and Gelatt, C. D. and Vecchi, M. P., 1983. Optimisation by simulated annealing. Science, 220, 671–680.

[15] Larsen, A., 2000. The Dynamic Vehicle Routing Problem. Department of MathematicalModelling, Technical University of Denmark.

[16] Löbel, A., 1998. Vehicle Scheduling in Public Transit and Lagrangian Pricing. Management Science, 44, 1637-1649.

[17] Lund, K. and Madsen, O. B. G. and Rygaard, J. M., 1996. Vehicle Routing Problems with Varying Degrees of Dynamism. Tech. rep., The Department of Mathematical Modelling, Technical University of Denmark.

[18] Martello, S. and Toth, P., 1990. Knapsack Problems: Algorithms and

Computer Implementations. Wiley, New-York.
[19] Tsang, E, 1993. Foundations of Constraint Satisfaction. Academic Press.
[20] Wren, A., 1998. Heuristics Ancient and Modern: Transport Scheduling Through the Ages. Journal of Heuristics, 4 (1), 87–100.
[21] Wren, A. and Fores, S. and Kwan, A. S. K. and Kwan, R. S. K. and Parker, M. and Proll, L. G., 2003. A Flexible System for Scheduling Drivers. Journal of Scheduling, 6 (5), 437–455.
[22] Wren, A. and Gualda, N. D. F., 1999. Integrated Scheduling of Buses and Drivers. Computer-Aided Transit Scheduling, In: Wilson, N. H. M. (Ed.), Springer Verlag, Berlin, 155–176.