TRANSPORTATION SCIENCE
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 0041-1655 | EISSN 1526-5447 | 00 | 0000 | 0001

INFORMS
DOI 10.1287/xxxx.0000.0000
© 0000 INFORMS

Randomized local search for real-life inventory routing

Thierry Benoist, Frédéric Gardi, Antoine Jeanjean
Bouygues e-lab, 40 rue Washington, 75008 Paris,
{tbenoist,fgardi,ajeanjean}@bouygues.com

Bertrand Estellon
Laboratoire d'Informatique Fondamentale - CNRS UMR 6166,
Faculté des Sciences de Luminy - Université Aix-Marseille II,
163 avenue de Luminy - case 901, 13288 Marseille cedex 9,
bertrand.estellon@lif.univ-mrs.fr

In this paper, a new practical solution approach based on randomized local search is presented for tackling a real-life inventory routing problem. Inventory routing refers to the optimization of transportation costs for the replenishment of customers' inventories: based on consumption forecasts, the vendor organizes delivery routes. Our model takes into account pickups, time windows, drivers' safety regulations, orders and many other real-life constraints. This generalization of the vehicle routing problem was often handled in two stages in the past: inventory first, routing second. On the contrary, a characteristic of our local-search approach is the absence of decomposition, made possible by a fast volume assignment algorithm. Moreover, thanks to a large variety of randomized neighborhoods, a simple first-improvement descent is used instead of tuned, complex metaheuristics. The problem being solved every day with a rolling horizon, the short-term objective needs to be carefully designed in order to ensure long-term savings. To achieve this goal we propose a new surrogate objective function for the short-term model, based on long-term lower bounds. An extensive computational study shows that our solution is effective, efficient and robust, providing long-term savings exceeding 20 % on average compared to solutions built by expert planners or even a classical urgency-based constructive algorithm. Confirming the promised gains in operations, the resulting decision support system is progressively deployed worldwide.

Key words: logistics; real-life inventory routing; decision support system; randomized local search; high-performance algorithm engineering.
History:

Inventory routing problems (IRP) integrates two classical logistic problems over a planning horizon: inventory management and vehicle routing. The main difference with vehicle routing is that the vendor monitors the customers' inventories, deciding when and how much each inventory should be replenished by routing vehicles. Based on consumption forecasts for each customer, a solution is a set of routes visiting customers and delivering a certain amount of product at each of these stops. In addition to routing constraints, stockouts must be avoided, which means that the quantity of product stored at each customer should stay above a certain safety level. The economic function to minimize is the costs of the routes.

In this paper, a real-life IRP is addressed. Having outlined the main features of the IRP model (Section 1), the contributions of the paper are emphasized with respect to prior works (Section 2). Based on lower bounds on delivery costs, a new surrogate objective is proposed for modeling the short-term problem, improving significantly the long-term optimization (Section 3). Then, an original local-search approach is presented which tackles frontally this short-term problem (Section 5), relying on a large variety of randomized neighborhoods and a fast volume assignment algorithm. An extensive computational study (Section 6) demonstrates that our solution provides long-term savings exceeding 20 % on average compared to solutions built by expert planners or even to a classical urgency-based constructive algorithm (Section 4).

1. The inventory routing model

For the sake of concision and readability, the problem is not completely and formally described, but its main characteristics are outlined. Comparisons with prior works will be developed in the next section.

A product is produced by the vendor's plants and consumed at customers' sites. Both plants and customers store the product in tanks. Reliable forecasts of consumption (resp. production) at customers (resp. plants) are available over a discretized short-term horizon. The inventory of each customer must be replenished by tank trucks so as to never fall under its safety level. The transportation is performed by vehicles formed by coupling three kinds of heterogenous resources: drivers, tractors, trailers. Each resource is assigned to a depot. Thus, the size of an instance of the problem is essentially defined by the number of points (depots, customers, plants), the number of resources (drivers, tractors, trailers), and the number of time steps for which are defined consumptions/productions over the horizon. Now, a solution to the problem is given as a set of shifts. A shift is defined by: the depot (from which it starts and ends), the vehicle (driver, tractor, trailer), its starting date, its ending date, and the chronological-ordered list of performed operations. Then, an operation is defined by the site where the operation takes place, the quantity delivered or loaded, its starting date (also called arrival date) and its ending date (also called departure date). Thus, the inventory levels for customers, plants and trailers can be computed from the quantities delivered or loaded during the shifts.

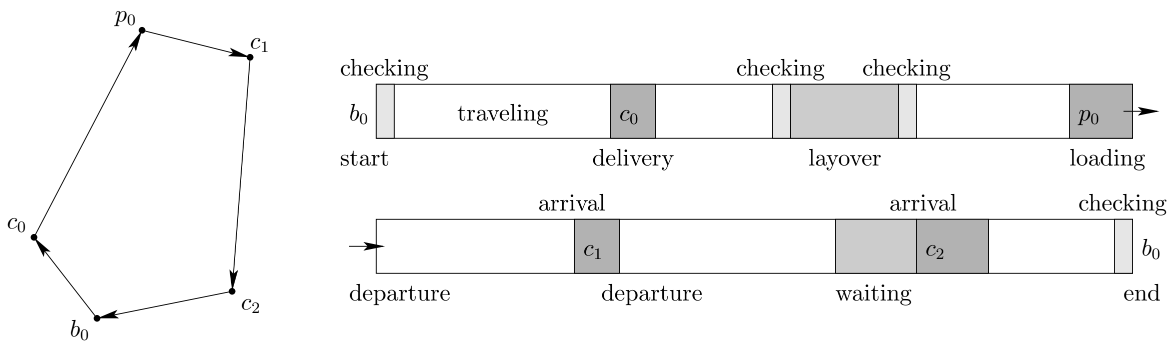

Figure 1 Two views of the shift $s = (b_0, c_0, p_0, c_1, c_2, b_0)$: the route and the schedule. Some of the notions described as real-life aspects are illustrated on the figure.

The constraints on shifts are called *routing constraints*. A shift must start from the depot which are located the resources composing the vehicle and must end by returning to this one. Some triplets of resources are not admissible (due to driving licences, for example). A resource can be used only during one of its availability time windows. The working and driving times of drivers are limited; as soon as a maximum duration is reached, the driver must take a layover with a minimum duration (legal rules and regulations). In addition, the duration of a shift cannot exceed a maximal value depending on the driver. Checking operations must be performed at the start and at the end of any shift, as well as before and after any layover. Traveling distance and time are given for each pair of points (depots, plants and customers). Instead of using a single time matrix, several matrices are defined depending on the average speeds of the different kinds of tractors. The resulting matrices are not necessarily symmetric, but are assumed to satisfy the triangular inequality. The sites visited along the tour must be accessible to the resources composing the vehicle (special skills or certifications are required to work on certain sites). The date of pickup/delivery must be contained in one of the opening time windows of the visited site. Here the duration of an operation does not depend on the delivered or loaded quantity; this duration is fixed in function of the site where the operation is performed, the resulting approximation being covered by the uncertainties lying on the traveled times. Two graphical views of a shift are illustrated on Figure 1.

Inventory constraints can be modeled as a flow network. Two kinds of inventories have to be managed: tanks of sites (customers and plants) and trailers. In any case, the quantity in a storage must remain between zero and its capacity. For a customer c, the tank level $l(c,h)$ at each time step h is equal to the tank level at the previous time step $h-1$, minus the forecasted consumption $F(c,h)$ over h, plus all the deliveries performed over h. The quantities delivered to customers must be positive (loading is forbidden at customers). Hence, the inventory dynamics at any time step h for customer c can be formally written as

$$\begin{cases} l(c,h) = l(c,h-1) - F(c,h) + \sum\limits_{o \in O(c,h)} q(o) \\ \text{if } l(c,h) < 0, \text{ then } l(c,h) = 0 \end{cases}$$

with $q(o)$ the quantity delivered during an operation o and $O(c,h)$ the set of operations performed at site c whose starting date belongs to time step h. Forbidding stockouts corresponds to force $l(c,h)$ to be greater than the safety level for each customer c and each time step h. The same formula applies for plants, if the forecasted productions and the loading quantities have negative values (delivery is forbidden at plants). However, for plants, when the above formula yields a quantity larger than the tank capacity, the result is limited to the capacity. These overflows are not penalized, because production aspects are assumed not to be managed in this model. For trailers, the inventory dynamics is much simpler since operations performed by a trailer cannot overlap. Hence the quantity in a trailer is not defined for each time step but after each of its operations. Starting at an initial level, this quantity is merely increased by loadings and decreased by deliveries.

This "forecasting-based resupply" model must be combined in practice with the classical "order-based resupply": the possibility for customers of issuing an order (to deal with an unexpected increase of their consumption, for example), specifying the desired quantity and the time window in which the delivery must be done. Some customers may use this possibility occasionally, while others choose to work in pure order-based resupply mode (without forecasting). But for the sake of concision, order management will not be deeply discussed in this paper.

The objective of the vendor *over the long term* (more than 90 days) is to minimize the total cost of shifts. The cost of a shift depends on the working duration (driver costs) and on the traveled distance (tractor and trailer costs), but also on the number of deliveries, loadings, and layovers appearing during the shift. This cost is usually divided by the total delivered quantity in order to increase the readability of this economic indicator. The cost per ton ratio (or miles per ton when costs are approximated by distances) is widely used in the industry and in the academic literature. This ratio $LR = SC/DQ$ is called *logistic ratio* throughout the paper, with SC the total shift cost and DQ the total delivered quantity over the considered horizon (with the exception that if $DQ = 0$, then $LR = 0$). On the other hand, reliable forecasts (for both plants and customers) are available over a 15-days horizon. Thus, shifts are operationally planned day after day with a *short-term rolling horizon of 15 days*. It means that each day, a distribution plan is deterministically built for the next 15 days, but only shifts starting at the current day are fixed.

Large-scale instances have to be tackled within short computing times. A geographic area can contain up to 1500 customers, 50 sources, 50 depots, 100 drivers, 100 tractors, 100 trailers. All temporal data have to be managed in continuous time, except for consumptions of customers (resp. productions of plants) which are discretely represented. Concretely, all dates and durations are expressed in minutes (on the whole, the short-term planning horizon counts 21600 minutes); the inventory dynamics for plants and customers are computed with time steps of one hour (because forecasts are computed with this accuracy). The execution time for computing a short-term planning is limited to 5 minutes on standard computers.

2. Prior works and contributions

Since the seminal work of Bell et al. (1983) on a real-life inventory routing problem, a vast literature has emerged on the subject. In particular, a long series of papers was published by Campbell et al. (1998, 2002), Campbell and Savelsbergh (2004a), Savelsbergh and Song (2007a,b, 2008), motivated by a real-life problematic encountered in the industry. However, in many companies, inventory routing is still done by hand or supported by basic softwares, with rules like: serve "emergency" customers (that is, customers whose inventory is near to run out) using as many "full deliveries" as possible (that is, deliveries with quantity equal to the trailer capacity or, if not possible, to the customer tank capacity). For more references, the interested reader is referred to the recent papers by Savelsbergh and Song (2007a, 2008), which give a comprehensive survey of the research done on the IRP over the past 25 years.

Below are emphasized the three contributions of the paper with respect to prior works:

- a very realistic model in terms of constraints, costs, and scale;
- a new surrogate objective function for dealing with the rolling horizon process;
- a pure and direct local-search heuristic, based on a fast volume assignment algorithm.

2.1. A real-life model

The problem addressed here is very close to the one treated by operational planners. To our acquaintance, such broad inventory routing problems have been rarely addressed in the operations research literature. Indeed, many real-life features described here have not been treated in past studies, allowing a more global and accurate optimization of the replenishment logistics. Some of these features have been reported as important practical issues in the survey by Campbell et al. (1998). First, our inventory routing model integrates both kinds of resupply: forecasting-based and order-based. Besides, several subproblems related to the scheduling of shifts and the allocation of resources to shifts become computationally hard in the present case. Another interesting feature, enabling to go further in logistic optimization while making the problem harder, is what is called "continuous moves" in Savelsbergh and Song (2007a, 2008) or "satellite facilities" in Bard et al. (1998). The vehicles can arbitrarily load or deliver some product along their routes, and loadings can be done at multiple plants. Moreover, when a driver reaches its working or driving time limit, he can continue his route after a layover. This allows to design shifts spanning several days and covering huge geographic areas. Then, the expected forecasts of consumption for customers (resp. production for plants) are given for each hour on a 15-days horizon, allowing nonlinear consumptions (resp. productions). Here forecasts are assumed to be reliable, inducing a deterministic optimization problem (contingencies on the customer consumption are considered to be covered by the defined safety level). Customers (resp. plants) may have different consumption (resp. production) profile, asking several deliveries (resp. pickups) per day or only one per month. Finally, a popular and sensible economic function, used by Campbell et al. (1998, 2002), Savelsbergh and Song (2007a, 2008), is to maximize the volume per mile over the long term, obtained by dividing the total quantity delivered to all customers by the total distance traveled. Instead of the sole traveled distance, we take into account the actual cost of the routes thanks to a precise modeling of the cost of each shift in function of its traveled distance, its traveled time, its number of loadings, its number of deliveries, and its number of rests. The resulting generalized objective is the minimization of the cost per unit of delivered product, called logistic ratio.

Note that one feature generally addressed in the IRP literature (e.g. Campbell et al. 1998, 2002, Savelsbergh and Song 2007a, 2008) is not included in our IRP model: loading or delivery duration depending on the quantity. Indeed, fixed-duration loadings and deliveries depending on sites were judged sufficient to approximate reality (full loadings/deliveries are performed in almost half an hour), because several other approximations making this detail negligible are done about temporal aspects due to real-life uncertainties (in particular about traveled times). Nevertheless, our solution could be adapted to manage this feature without significantly affecting its performance.

2.2. A new surrogate objective function

While the IRP goal is to optimize distribution costs over the long term, forecasts are usually available for a short horizon and continuously updated. Therefore the problem is usually solved in a rolling horizon framework. Here a short term planning is build for 15 days and only shifts starting the first day are fixed. The next day, a new planning is built including one more day in the horizon. As mentioned by Campbell et al. (1998, 2002), the first difficulty arising in modeling IRP is to define appropriate short-term objectives leading to good long-term solutions. In fact, a direct minimization of costs would lead to deferring as many deliveries as possible to later planning periods. This rolling horizon issue has been studied in Dror and Ball (1987), where incentives are introduced for the amount delivered, based on an estimation of the long term savings incurred by anticipated deliveries. Our contribution on this topic is to introduce a new surrogate objective for short-term optimization (here done over a 15-days horizon) ensuring long-term improvements. This surrogate objective, which shall be detailed later in the paper, is based on lower bounds for the logistic ratio (this extends observations made by Savelsbergh and Song (2007b) on performance measurement). Computational experiments with real-life data show that significant gains are obtained in the long run by optimizing this short-term surrogate objective, compared to a direct short-term minimization of the logistic ratio.

2.3. A direct local-search approach

To our knowledge, the sole papers describing practical solutions for similar problems are the ones described by Campbell et al. (2002), Campbell and Savelsbergh (2004a), Savelsbergh and Song (2007a, 2008). Before presenting our solution approach, we outline the ones implemented by Campbell et al. (2002), Campbell and Savelsbergh (2004a) for solving the single-plant IRP, and by Savelsbergh and Song (2007a, 2008) for solving the multiple-plant IRP. The solution approaches described by Campbell et al. (2002) and Campbell and Savelsbergh (2004a) are the same in essence; because integrating additional realistic constraints, the single-plant IRP addressed by Campbell and Savelsbergh (2004a) is more complex than the one by Campbell et al. (2002). The method developed by the authors is deterministic and proceeds in two phases. In the first phase, it is decided which customers are visited in the next few days, and a target amount of product to be delivered to these customers is set. In the second phase, vehicle routes are determined taking into account vehicle capacities, customer delivery windows, drivers restrictions, etc. The first phase is solved heuristically by integer programming techniques, whereas the second phase is solved with specific insertion heuristics (Campbell and Savelsbergh 2004c), as done for vehicle routing problems with time windows by Solomon (1987).

In Savelsbergh and Song (2007a, 2008), the authors develop two approaches for solving the multiple-plant IRP. Many realistic features taken into account in Campbell and Savelsbergh (2004a) are relaxed in the model addressed by the authors. In particular, simple resources are considered (that is, a vehicle is reduced to a trailer) allowing an integer multi-commodity flow formulation of the problem. The first approach (Savelsbergh and Song 2007a) is based on an insertion heuristic which delivers customers ordered by urgency (that is, the time remaining before the first stockout) while minimizing stockout and transportation costs. This approach is declined into three greedy algorithms: a basic one where insertions are only performed at the end of shifts, a enhanced one where insertions can be performed at any point in the shift after the last pickup, and a randomized enhanced one where the latter is embedded into a greedy randomized adaptive search procedure (Feo and Resende 1995). Then, a postprocessing is performed using linear programming for maximizing delivered quantities on the resulting shifts (in order to maximize the volume per mile). The second approach (Savelsbergh and Song 2008) consists in solving heuristically the integer multi-commodity flow program (by using customized integer programming techniques). Since such computational requirements are too large for a practical use, the authors use the integer program for exploring large neighborhoods in a local-search scheme. It consists in re-optimizing the schedules of two vehicles in the planning by solving the integer program with the other schedules fixed. In this way, all pairs of vehicles are re-optimized iteratively.

For the resolution of the short-term planning problem with the surrogate objective, a local-search heuristic is presented in this paper, whose design and engineering follows the "three-layers" methodology introduced by Estellon et al. (2009) and successfully implemented for solving other large-scale combinatorial problems arising in business and industry (Estellon et al. 2006, 2008, 2009). The originality of our local-search approach is to be *pure* (no metaheuristic, no hybridization) and *direct* (no decomposition of the problem: the 15-days planning is directly optimized). The absence of decomposition is a strength of our approach. Indeed, most previous works on the subject proceed in two stages: inventory management first and vehicle routing second. On the contrary, our local-search heuristic handles the whole problem and performs transformations on the solution impacting both deliveries (dates and volumes) and routes (paths, resources, etc.). The performance of our approach relies on: 1) the use of a large variety of randomized small neighborhoods, allowing to reach high-quality local optima without the help of metaheuristics; 2) the engineering of fast evaluation routines, in particular an incremental approximation algorithm for volume assignment which is a thousand times faster than exact flow algorithms. Note that a local-search approach is outlined by Lau et al. (2002) for solving an IRP with time windows, but their solution remains based on a decomposition of the problem (distribution and then routing). For more details on local-search techniques and their applications in combinatorial optimization, the reader is referred to the book edited by Aarts and Lenstra (1997).

The experiments reported in the related works Bell et al. (1983), Campbell et al. (2002), Campbell and Savelsbergh (2004a), Savelsbergh and Song (2007a, 2008) show savings up to 10 % over the long run (with computation times of several minutes), compared to solutions obtained by a greedy algorithm based on the rules of thumb commonly used in practice (like the one cited in introduction of this section). The extensive computational study presented at the end of the paper demonstrates that our solution is effective, efficient and robust, providing long-term savings exceeding 20 % on average, compared to solutions computed by expert planners or even to a classical urgency-based constructive heuristic. An abstract of this work was previously published by Benoist et al. (2009).

3. The short-term surrogate objective

One of the main difficulties encountered in IRP is to take short-term decisions ensuring long-term improvements. This is the case with most problems handled with a rolling horizon approach. In other words, a surrogate short-term objective needs to be defined in order to optimize the economic function over the long term. Obviously, focusing on the minimization of total costs would lead to delivering only customers for which a shortage is forecasted over the short-term horizon. Optimizing the logistic ratio allows better anticipations but does not lead necessarily to long-term optimal solutions. For example, assume that a faraway customer has no stockout over the next days. Scheduling a delivery for this customer would probably increase the logistic ratio, and will be discarded. But such short-term decisions may be highly suboptimal in the long run, especially if some near-optimal deliveries are possible over these next days due to the availability of resources. This lack of anticipation motivates the proposal of a new surrogate objective function that can be summarized into the following rule: "never put off until tomorrow what you can do optimally today". This short-term goal shall be to minimize the global extra cost per unit of delivered product, compared to the optimal logistic ratio LR^*. Denote by $LR^*(c)$ the optimal logistic ratio for delivering the customer c and then by

$$SC^*(s) = \sum_{\substack{customer\, c \\ delivered\ over\ s}} LR^*(c) \times q(c)$$

the optimal cost of the shift s according to the quantities delivered at each customer over s. Then, the surrogate logistic ratio LR' is defined as:

$$LR' = \frac{\sum_s (SC(s) - SC^*(s))}{DQ}$$

Then, it requires to tackle another problem: the computation of lower bounds of $LR^*(c)$ for each customer c. These lower bounds are computed as follows. A trip is defined as a subpart of a tour (see Figure 2): it is a sequence of visits starting at a plant (or a depot), delivering one or more customers, and finishing at a plant (or a depot). In other words, a trip in a shift corresponds to a time interval whose left endpoint (resp. right endpoint) is the starting date from the plant or the depot (resp. the starting date from the plant or the depot visited in the next trip). Then, the cost of a shift can be decomposed according to its trips, in such a way that the cost of a trip corresponds to the costs (distance, time, deliveries, loadings, layovers) accumulated over the corresponding time interval. Besides, the cost of each trip can be dispatched to visited customers proportionally to the delivered quantities. For each customer c, a lower bound $LR_{\min}(c)$ is obtained by dividing the cost of the cheapest trip visiting c by the maximum capacity of a trailer able to perform this trip. Since the distance and time matrices satisfy the triangular inequality, the cheapest trip consists in visiting solely the customer c. Consequently, $LR_{\min}(c)$ is computed in $O((B+P)^2)$ time for each customer c, with B the number of depots and P the number of plants.

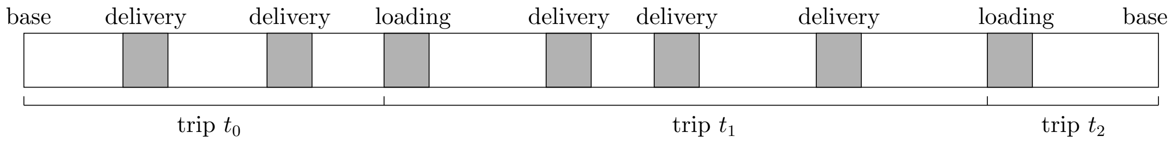

Figure 2 The trips of a shift.

These lower bounds $LR_{\min}(c)$ are used in the definition of the surrogate objective function LR'. A global lower bound of the logistic ratio LR^* can be derived from these lower bounds, but this topic is outside of the scope of the present paper.

4. Urgency-based constructive heuristic

In order to quickly build an initial solution, a constructive heuristic was designed, based on a classical urgency approach. The goal of this algorithm is to avoid stockouts. Basically, it repeatedly picks the next stockout and tries to create a new delivery for this customer. The deadline of a demand (stockout) is defined as the latest start of a shift that would reach the customer on time to perform the desired delivery, taking travel time and opening hours into account.

```
Algorithm GREEDY;
Input: an instance $I$ of IRP;
Output: a solution $S$ to IRP (namely a set of shifts);
Begin;
  $S \leftarrow \emptyset$;
  initialize the set $D$ of demands with stockouts for all customers;
  while $D$ is not empty do
    select the demand $d \in D$ with earliest deadline;
    create a cheapest delivery $o$ to satisfy $d$;
    if $o$ exists then
      insert $o$ into a shift in $S$ (possibly creating a new shift);
      compute the next stockout after the delivery $o$ and update the deadline of $d$ accordingly;
```

```
        else remove $d$ from $D$;
    return $S$;
End;
```

At each step of the algorithm, the newly created delivery can be either appended at the end of an existing shift or included in a new shift. Our goal is to perform the insertion minimizing the increase of the surrogate logistic ratio. In the first case, the extension of a shift can be made impossible due to accessibility or resources constraints (for example, the resulting duration of this shift may extend the maximum allowed amplitude). For each existing shift, this feasibility is tested in constant time. However, inserting a loading operation can be required for refilling the trailer before performing the delivery, in which case all plants are tested. Therefore, this stage runs in $O(|S|P)$ time, with $|S|$ the number of shifts returned by the greedy algorithm and P the number of plants. In the second case, all depots and all possible triplets of resources are considered. Here again, all plants are considered if a loading is needed. The worst-case time complexity of this enumeration is in $O(BRP)$, with B the number of depots and R the number of triplets of resources (drivers, tractors, trailers). But in effect, this running time can be reduced by cutting strongly the search tree, in particular once a feasible shift has been found.

The choice of the delivery date impacts the delivered volume since the available space in the customer tank increases with time. On the other hand, packing the shifts to the left tends to optimize resources utilization. Our tradeoff consists in favoring large deliveries as a first criterion, breaking ties by preferring early dates. In other words, we prefer to deliver later (but before the stockout date) if it allows increasing the delivered quantity, but between two dates allowing to empty the trailer we choose the earliest one. Each time a delivery is created, the inventory levels for this customer are updated and its next stockout date is computed.

Never backtracking on decisions taken on dates and quantities, this urgency-based insertion heuristic runs fast in practice. Even if the local-search heuristic described in the next section is able to start from an empty set of shifts, the use of the initial solution obtained by this constructive algorithm yields a significant speed-up in the convergence toward high-quality solutions (in particular, when finding a solution without stockout is hard).

5. High-performance local search

In this section, the main ingredients of the local-search heuristic are detailed. The exposition follows the "three-layers" methodology by Estellon et al. (2009) for designing local-search algorithms: heuristic & search strategy, transformations, incremental evaluation machinery.

5.1. Heuristic & search strategy

The heuristic is divided into two optimization phases: first we reach a feasible solution, then we optimize the cost of this solution. Indeed, when finding a feasible solution is not trivial, a common modeling practice is to relax some constraints by introducing appropriate penalties. Such relaxations are especially needed in the context of heuristic solution approaches. In the present case, we relax the no-stockout constraints as detailed below. In the first phase, the only objective is the maximization of our feasibility measure, regardless of costs. As soon as a feasible solution is found, we switch to the second phase (optimization of the economic function) during which only feasible solutions are visited.

During the feasibility stage, the no-stockout constraints are removed and a penalty SO is introduced, counting for each customer the number of time steps during which the above inequality is violated (quantity in tank is smaller than the safety level). In other words, the feasibility stage consists in minimizing the number of violated no-stockout constraints. This stage is necessary for finding feasible solutions, since the constructive heuristic described in the previous section may not fix all stockouts. It also ensures that the software always returns a (possibly infeasible) solution. Indeed, presenting an infeasible solution to the user allows him to analyze the causes for this infeasibility. For instance, if stockouts are avoided for all customers but one, he can focus on this customer and detect that no available trailer is allowed to visit this site.

Below is outlined the skeleton of the whole heuristic, which is a simple first-improvement randomized descent. We insist on the fact that no metaheuristic is used, thus avoiding the need for many tuning parameters.

```
Algorithm RANDOMIZED-DESCENT;
Input: an instance $I$ of IRP;
Output: a solution $S$ to IRP;
Begin;
  $S \leftarrow$ GREEDY($I$);
Stockout optimization:
  while $SO > 0$ and time limit is not reached do
    choose randomly a transformation $T$ in the pool $\mathcal{T}_{SO}$;
    evaluate the result ($SO_{\text{new}}$) of the application of $T$ to $S$;
    if $SO_{\text{new}} \leq SO$ then commit $T$; else rollback $T$;
Logistic ratio optimization:
  while time limit is not reached do
    choose randomly a transformation $T$ in the pool $\mathcal{T}_{LR}$;
    evaluate the result ($SO_{\text{new}}$, $LR'_{\text{new}}$) of the application of $T$ to $S$;
    if $SO_{\text{new}} = 0$ and $LR'_{\text{new}} \leq LR'$ then commit $T$; else rollback $T$;
  return $S$;
End;
```

Thus, the heuristic is divided into two optimization phases: the first one consists in minimizing the number of stockout time steps (SO) regardless of costs, and the second one consists in optimizing the objective related to logistic ratio (LR') while preserving $SO = 0$. Accepting solutions with equal cost at each optimization phase is crucial for ensuring the diversification of the search and thus the convergence toward high-quality solutions. For each optimization phase, the transformation to apply is chosen randomly with equal probability over all transformations of the pool (improvements being marginal, further tunings with non-uniform distribution have been abandoned to facilitate maintenance and evolutions). Note that in our implementation, the sign of $LR'_{new} - LR'$ is obtained by evaluating the expression

$$\frac{SC - SC^*}{DQ} - \frac{SC_{\text{new}} - SC^*_{\text{new}}}{DQ_{\text{new}}}$$

or equivalently $DQ_{\text{new}}(SC - SC^*) - DQ(SC_{\text{new}} - SC^*_{\text{new}})$ which avoids imprecisions due to floating-point arithmetic when the expression tends toward zero.

In presence of orders, the above heuristic needs to be enriched with a preliminary phase devoted to orders (as an explicit request of a customer, order satisfaction have the highest priority). Hence, our objective becomes in lexicographic order: order satisfaction, stockout satisfaction and finally, logistic ratio minimization. Our measure of order satisfaction is not directly the count of satisfied orders. Introducing for each order an intermediate state called “unsatisfied” between the states “missed” and “satisfied” was found to facilitate the convergence of the local search. An order is unsatisfied if an operation exists satisfying the time window of the order, but not its quantity. In this way, an order is satisfied (resp. missed) when both the dates and the quantity are respected by at least one operation (resp. by no operation). A transformation is accepted when it increases the number of satisfied orders or when it leaves this number unchanged, without increasing the number of unsatisfied orders. Together with specialized transformations focused on orders, this phase allows to handle problems where “forecasting-based resupply” and “order-based resupply” are mixed. For the sake of concision, this variant of the problem will not be evoked in the remaining of this paper nor in experimental results.

5.2. The transformations

The transformations are classified into two categories: the first ones work on operations, the second ones work on shifts. Having introduced the different transformations, their main instantiation shall be described. An instantiation corresponds to the way the objects modified by the transformation are selected. While defining orthogonal transformations (that is, transformations inducing disjoint neighborhoods) enables to diversify the search and then reach better-quality solutions, specializing transformations according to specificities of the problem (because random choices are not the most appropriate in all situations) enables to intensify the search and then speed up the convergence of the heuristic.

The transformations on operations are grouped into the following types: insertion, deletion, ejection, move, swap (see Figure 3). Two kinds of *insertion* are defined: the first kind consists in inserting an operation (pickup or delivery) into an existing shift; the second consists in inserting a pickup followed by a delivery into a shift (the inserted plant is chosen to be one of the nearest from the inserted customer). The *deletion* consists in deleting a block of operations (that is, a set of consecutive operations) in a shift. An *ejection* consists in replacing an existing operation by a new one on a different site. The *move* transformation consists in extracting a block of operations from a shift and reinserting it at another position. Two kinds of moves are defined: moving operations from a shift to another one, or moving operations inside a shift. A *swap* exchanges two different blocks of operations. As for moves, several kinds of swaps are defined: the swap of blocks between shifts, the swap of blocks inside a shift, or the "mirror" which consists in a chronological reversal of a block of operations in a shift. The mirror transformation corresponds to the well-known 2-opt improvement used for solving traveling salesman problems (see Aarts and Lenstra (1997) for more details).

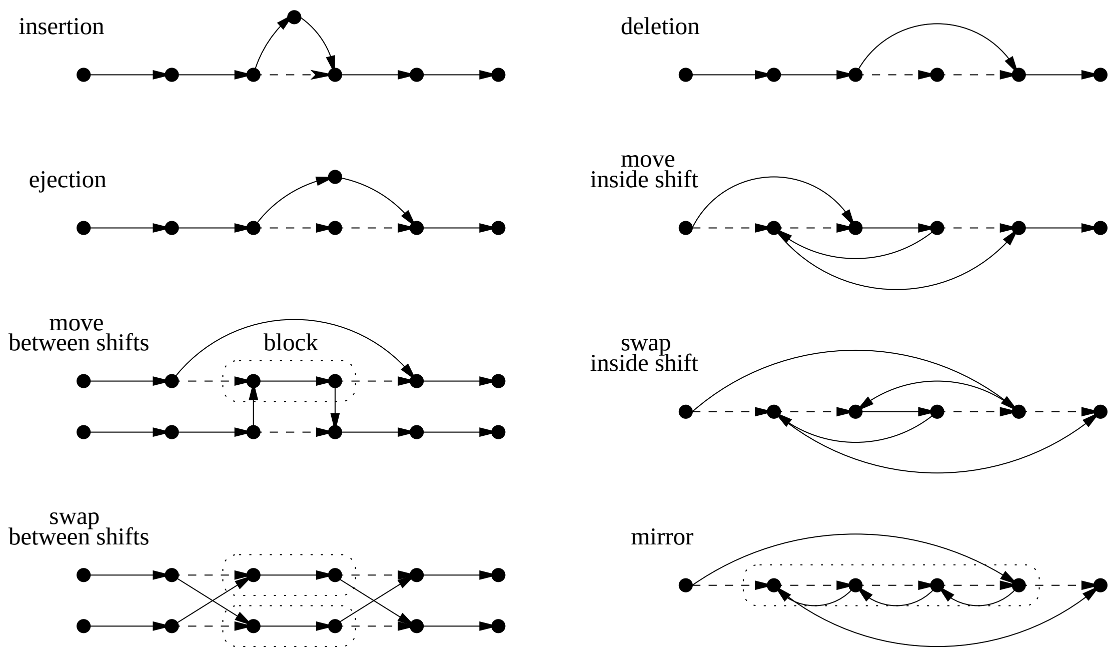

Figure 3 The transformations on operations.

Note. Original tours are given by straight arcs, dashed arcs are removed by the transformation, curved and vertical arcs are added by the transformation.

The transformations on shifts are grouped into the following types: insertion, deletion, rolling, move, swap. As for operations, two kinds of *insertion* are defined: insertion of a shift containing one operation (pickup or delivery), insertion of a shift with a pickup followed by a delivery. *Deletion* consists in removing an existing shift. The *rolling* transformation translates a shift over time. The *move* consists in extracting a shift from the planning of some of its resources and reinserting it into the planning of other ones (such a transformation allows to change some of the resources of the shift and its starting date). The *swap* is defined similarly: the resources of the shifts are exchanged and their starting dates can be translated over time. The *fusion* of two consecutive shifts into one new shift as well as the *separation* of one shift into two new ones are also available.

Now, these transformations are declined from different ways. The first option concerns the maximal size of blocks for transformations where blocks of operations are involved. In this way, more generic transformations are defined allowing a larger diversification if needed: the (k,l)-ejection which consists in replacing k existing operations by l new ones on different sites, the k-move which consists in moving a block of k operations, the (k,l)-swap which consists in exchanging a block of k operations with a block of l operations, or the k-mirror which consists in reversing a block of k operations.

Then, the second option allows to specialize some transformations. These derivations involve the choice of the sites affected by the transformations. For example, inserting a delivery serving a customer without expected stockout in the considered horizon is not interesting when minimizing the number of stockouts. In the same way, exchanging two operations which are performed on sites which are very distant is unlikely to succeed when optimizing the logistic ratio. Several derivations have been designed, which differ slightly from one transformation to another. Here are given the two main derivations, essentially used when inserting/ejecting operations or inserting/rolling shifts: "stockout" which places the delivery so as to solve a stockout, "nearest" where the customers to be inserted or exchanged are chosen among the nearest ones.

The third option corresponds to the direction used to recompute all the dates of the modified shifts: backward over time by considering the ending date of the shift as fixed, or forward over time by considering its starting date as fixed. This option is available for all transformations, except the deletion of shifts. For the transformations modifying two shifts at once (for example, move operations between shifts), this results in four possible instantiations: backward/backward, backward/forward, forward/backward, forward/forward. Finally, the fourth option allows to augment the number of operations whose quantities are modified during the volume assignment. Recomputing operation quantities during the volume assignment increases its running time but allows repairing stockouts possibly introduced by the transformation, increasing the acceptation rate of the transformations (more details are given in the next section about volume assignment).

The reader shall note that no very large-scale neighborhood is employed. Roughly speaking, the neighborhood explored here has a size $O(n^2)$ with n the number of operations and shifts in the current solution, but the constant hidden by the O notation is large. The number of transformations in $\mathcal{T}_{SO}$ and $\mathcal{T}_{LR}$ used respectively in optimization phases SO and LR are of 49 and 71. As evoked previously, the random selection of transformations is simply performed with a uniform distribution.

5.3. Incremental evaluation machinery

Finally, the evaluation kernel of the local search is outlined. Playing a central role in the efficiency of the local-search heuristic, only the evaluation procedure is detailed here. For each transformation, this procedure follows the same process:

1. *Transforming the current solution.* The transformation modifies shifts and operations. Resource/resource and resource/site compatibilities are checked.

2. *Scheduling shifts.* Modified shifts are rescheduled in order to compute the new dates. All routing constraints (time windows, driver regulations, etc.) are taken into account.

3. *Assigning volumes.* Delivered or loaded quantities are recomputed for a limited number of sites including the modified ones. Inventories are updated, and the new number of stockouts SO is computed.

Roughly speaking, the objective of the scheduling routine is to build shifts with smallest costs, whereas the volume assignment tends to maximize the quantity delivered to customers. Even approximately, it leads to minimize the surrogate logistic ratio.

5.3.1. Scheduling shifts. The transformations modify some shifts in the current solution (at most two actually). When a shift is impacted by a transformation (for example, an operation is inserted into the shift), the starting and ending dates of its operations must be computed anew. Consider the shift $s = (o_1, \ldots, o_n)$ and assume that an operation $\bar{o}$ is inserted into s between operations i and j. The resulting shift $\bar{s}$ is now composed of operations $(o_1, \ldots, o_i, \bar{o}, o_j, \ldots, o_n)$. Then, we have two possibilities: rescheduling dates forward or rescheduling dates backward. The forward (resp. backward) scheduling consists in fixing the ending date of o_i (resp. the starting date of o_j) in order to recompute the starting (resp. ending) dates of $(\bar{o}, \ldots, o_n)$ (resp. $(o_1, \ldots, \bar{o})$). Here computing dates can be done without assigning volumes to operations, because the durations of operations do not depend on delivered/loaded quantities. Since computing dates backward or forward is made completely symmetric by representing shifts with doubly-linked lists, the discussion shall be reduced to the forward case.

More formally, we have to solve the following decision problem, called Shift-Scheduling: given a starting date for the shift $s = (o_1, \ldots, o_n)$, determine the dates of each operation such that the shift is admissible. Two equivalent optimization problems are: having fixed its starting date, build a shift with the earliest ending date or with the minimum cost. A similar problem, called Truckload-Trip-Scheduling, has been recently studied by Archetti and Savelsbergh (2007). This latter problem is more restricted in the sense that only one opening time window is considered for each location to visit and that the rest time must be equal to (not greater than) the legal duration. Archetti and Savelsbergh (2007) sketch a $O(n^2)$-time algorithm for solving the truckload trip scheduling problem, with n the number of locations to visit. For the sake of efficiency, a linear-time and space algorithm has been designed for solving heuristically the Shift-Scheduling problem.

```
Algorithm Schedule-Shift-Greedy;
Input: an instance of Shift-Scheduling;
Output: an admissible shift if any, null otherwise;
Begin;
  define an empty shift;
  for each location to visit do
    drive to next location (by taking rests as late as possible if needed);
    if waiting time is needed (due to opening time windows) then
      if rest time has been taken on current arc then
        lengthen one of the rests to absorb waiting time;
      else if a rest is needed (due to waiting time) or waiting time is larger than rest time then
        take a rest (absorbing additional waiting time if any);
      else
        wait for the opening of location;
    perform operation at next location and add it to the shift;
    if maximal amplitude of the shift is exceeded then return null (infeasibility);
  return the shift (feasibility);
End;
```

This algorithm is greedy in the sense that operations are chronologically set without backtracking. Each loop is performed in constant time and space (if rests are not stored explicitly) and the whole algorithm runs in $O(n)$ time and space. The correctness of the algorithm is ensured by

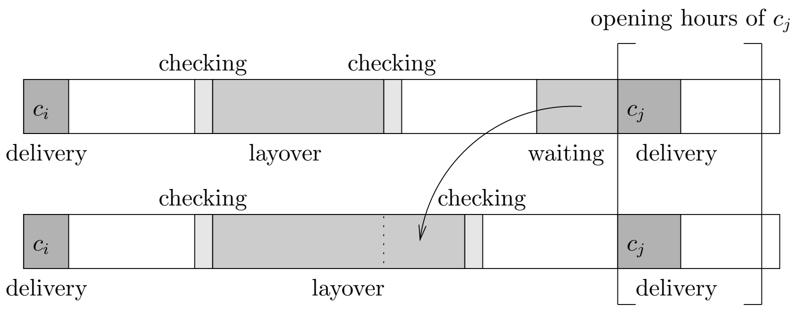

Figure 4 An example with waiting time converted into rest time.

construction. The key of the Shift-Scheduling problem is to minimize unproductive time over the shift. Thereby, the main idea behind the algorithm is to take rests as late as possible during the trip and to avoid waiting time due to opening time windows of locations as much as possible. Here we try to remove waiting time by converting it into rest time (see Figure 4), but only on the current arc, which is suboptimal. Indeed, the algorithm could be reinforced by trying to convert waiting time into rest time on previous arcs (as done by Archetti and Savelsbergh (2007)). But such a modification would lead to a quadratic-time algorithm, which is not desired here, while not guaranteeing optimality because of multiple opening time windows. On the other hand, we have observed that waiting time is rarely generated in practice since many trips are completed in a day or even half a day, ensuring optimality of the algorithm Schedule-Shift-Greedy in most cases. Note that to our knowledge, the complexity of the Shift-Scheduling problem remains unknown.

5.3.2. Assigning volumes. Having rescheduled modified shifts, we have to reassign quantities to impacted operations. Having fixed the dates of all operations, the problem consists in assigning volumes such that inventory constraints are respected, while maximizing the total delivered quantity over all shifts. A similar problem, called Delivery-Volume-Optimization, has been addressed by Campbell and Savelsbergh (2004b). In this problem, the authors consider only deliveries on routes and not loadings, but this one is complicated by the fact that the duration of an operation depends on the quantity delivered.

From the theoretical point of view, the present problem, called Volume-Assigning, is not so hard once observed that it can be formulated as a maximum flow problem (in a directed acyclic network). Then, this one can be solved in $O(n^3)$ time by using a classical maximum flow algorithm (Cormen et al. 2004, pp. 625–675), with n the number of operations. As mentioned previously, such a time complexity is not desirable here, even if guaranteeing an optimal volume assignment. Practically, naive implementations having a time complexity depending on the number H of time steps (360 in practice) are prohibited too; indeed, when the granularity becomes smaller than one day, the number of time steps exceeds largely the number of operations at a site (two per day in the worst case).

Thus, a $O(n \log n)$-time greedy algorithm has been designed to solve approximately the Volume-Assigning problem. The main idea behind the algorithm is simple: having ordered operations chronologically (that is, according to increasing starting dates), quantities are assigned to operations in this order following a greedy rule. Here we use the basic rule consisting in maximizing the quantity delivered/loaded at each operation, which is a good policy for minimizing the surrogate logistic ratio (this joins the ideas developed by Campbell and Savelsbergh (2004b)). Note that the chronological ordering is crucial for ensuring the respect of constraints related to inventory dynamics (flow conservation, capacity constraints). In graph-theoretical terms, the algorithm consists in pushing flow in the induced directed acyclic network following a topological order of the nodes (ensuring that no node is visited twice).

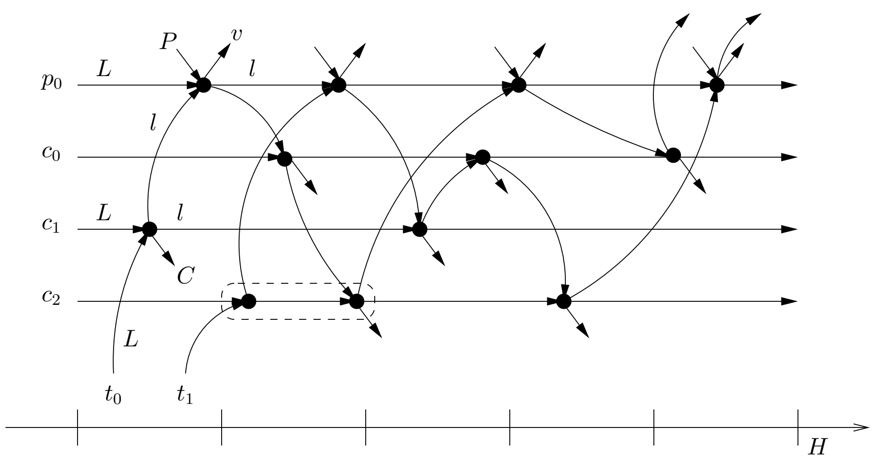

Figure 5 An example of flow network for assigning volumes.

Note. Operations are represented by nodes, input flows L correspond to initial levels for each inventory (trailers t_i, customers c_i, plants p_i), input flow C (resp. P) corresponds to consumption of customer c_1 (resp. production of plant p_0) over the time steps between the current operation and the previous one, flows l correspond to inventory levels between two operations, flow v allows an overflow at plant. Flows on arcs representing inventory levels are upper bounded by the capacity of the inventory; for customers, flows are also lower bounded by safety levels. Note that if some consecutive operations appear over the same time step (like the ones dotted around), input flows corresponding to consumption or production are cumulated at the last operation of this time step.

Because the number of operations may be large (as worst case in practice, one can imagine that a thousand customers must be delivered one time per day, leading to $n \geq 10\ 000$), a tradeoff must be found between the time complexity (even linear) and the quality of the volume assignment. To introduce flexibility on this point, the greedy algorithm has been designed for computing partial reassignments, from the minimal one to the complete one. The minimal reassignment consists in changing only the volumes on impacted operations (that is, operations whose starting dates are modified by the transformation); then, it suffices to tag as impacted some additional operations to expand the reassignment. This complicates notably the practical implementation of the greedy algorithm. Indeed, changing the quantity delivered at an operation is delicate since increasing (resp. decreasing) the quantity may imply overflows (resp. stockouts) at future operations. Then, determining the (maximum) quantity to deliver/load at each operation is not straightforward.

For each site p, denote by $\bar{n}_p$ the number of operations between the first impacted operation (that is, whose quantity can be modified by the transformation) in the chronological ordering and the last one over the horizon. If no operation is impacted at site p, then $\bar{n}_p = 0$. Hence, we define $\bar{n} = \sum_p \bar{n}_p$. When the set of impacted operations consist only in operations whose dates are modified by the transformation, one can observe in practice that $\bar{n} \ll n$, since each transformation modify at most two shifts (the number of sites visited by one shift is generally small). Consequently, it is important to provide algorithms whose running time is linear in $O(\bar{n})$, and not only in $O(n)$. Below is outlined an $O(\bar{n} \log \bar{n})$-time algorithm for assigning volumes. But before, more explanations are given on how the maximum deliverable quantity is computed (the maximum loadable quantity can be obtained in a symmetric way).

Denote by $l(c,i)$ (resp. $l(t,i)$) the level of customer c (resp. trailer t) before starting the operation i and by $qmax(c,i)$ the maximum quantity that can be delivered to customer c at operation i without inducing overflows until the end of the horizon. In this way, the deliverable quantity at operation i, denoted by $q(i)$, is upper bounded by $\min\{l(t,i), qmax(c,i)\}$. Then, this bound is reinforced in such a way that the quantity remaining in the trailer after a delivery is sufficient to avoid stockouts at customers visited by the shift until the next loading. Denote by $qmin(c,i)$ the minimum quantity to deliver at operation i to avoid stockout until the end of the horizon. Now, the minimum quantity $qmin(t,i)$ which must remain in the trailer t after operation i to avoid stockout later is obtained by summing $qmin(c,i)$ for all operations between the current one and the next loading. Then, we have

$$q(i) \leq \min\{l(t,i) - qmin(t,i), qmax(c,i)\}$$

Given the chronological-ordered list of operations for each trailer, customer and plant, all the data structures mentioned above can be computed in $O(\bar{n})$ time. Updating $l(c,i)$ (resp. $l(t,i)$) for any operation i is done by sweeping forward the operations delivering customer c (resp. performed by the trailer t). Then, updating $qmin(c,i)$ and $qmax(c,i)$ for any operation i is done by sweeping backward the operations performed at customer c (note that the consumption between two operations is obtained in constant time by storing cumulated consumptions over the horizon). Finally, computing $qmin(t,i)$ for any operation i is done by sweeping backward the operations of shifts performed by t. Below is given a sketch of algorithm.

```
Algorithm Assign-Volumes-Greedy;
Input: the set $\mathcal{E}$ of $\bar{n}$ impacted operations;
Begin;
  sort the set $\mathcal{E}$ chronologically;
  for each impacted customer $c$ do update $l(c)$, $qmin(c)$, $qmax(c)$;
  for each impacted trailer $t$ do update $l(t)$, $qmin(t)$;
  for each operation in $\mathcal{E}$ do
    assign the maximum deliverable/loadable quantity to the operation;
End;
```

A basic example is given for illustrating the reset of delivered quantities and the update of data structures. A customer has 4 scheduled operations $A = (4,2)$, $B = (10,5)$, $C = (14,8)$, $D = (18,12)$ with the first number denoting the time step to which the operation occurs and the second number denoting the delivered quantity. Here the horizon is composed of 24 time steps. The capacity C of the tank is 14 and the safety level S is 2. Table 1 gives for each time step h the forecasted consumption $F(h)$ and the resulting tank level $l(h)$. Then, values in data structures *qmin* and *qmax* are detailed. These values are computed backward over the horizon by applying the following recurrence:

$$qmin(h) = \max\{qmin(h+1), S - l(h) + F(h)\}$$

$$qmax(h) = \min\{qmax(h+1), C - l(h) + F(h)\}$$

Now, let us consider that operation B is impacted by a transformation: only the quantity of operation B can be modified. The delivered quantity of operation B is reset to zero. Then, for assigning a new quantity to operation B, only the values of column 10 have to be updated: $l(10)$ is decreased by 5, while $qmin(10)$ and $qmax(10)$ are increased by 5. In this way, the new (temporary) values are: $l(10) = 2$, $qmin(10) = 4$ and $qmax(10) = 6$. It means that a minimal quantity of 4 has to be delivered to avoid stockouts over the horizon (here time step 17 is critical) and a maximal quantity of 6 can be delivered without implying overflows (here time step 18 is critical). We insist on the fact that the other columns will only be updated if the transformation is eventually committed.

Table 1 **Data structures for assigning volumes.**

h	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
F	0	1	0	1	1	1	2	3	2	2	0	1	1	1	2	1	3	3	2	2	1	2	3	0
q	-	-	-	-	2	-	-	-	-	-	5	-	-	-	8	-	-	-	12	-	-	-	-	-
l	13	12	12	11	12	11	9	6	4	2	7	6	5	4	10	9	6	3	13	11	10	8	5	5
qmin	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-3	-3	-3	-3	-3	-3
qmax	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3	4	6	9	9

According to the previous discussion, the five data structures which serve to the calculation of the maximum deliverable/loadable quantity are updated in $O(\bar{n})$ time (recall that $\bar{n}$ is the number of impacted operations). Then, sorting the set $\mathcal{E}$ is done in $O(\bar{n}\log\bar{n})$ time in the worst case; in effect, the heapsort algorithm is used (see Cormen et al. (2004, pp. 121–137)). Finally, the loop is done in linear time, since the calculation of the maximum deliverable/loadable quantity requires only constant time by using the adequate data structures. Consequently, the whole algorithm runs in $O(\bar{n}\log\bar{n})$ time.

In theory, the greedy algorithm is far from being optimal. Figure 6 gives the smallest configuration for which the greedy algorithm fails to find an optimal assignment. On the other hand, two sufficient conditions hold for which the greedy assignment is optimal. The proofs are not detailed here because easy to establish. The first condition corresponds to the case where each customer is served at most once over the planning horizon. This condition is interesting because likely to be met in practice. The second condition corresponds to the case where each shift visits only one customer. For example, this condition is satisfied when customers have infinite storage capacity.

Experiments have been made for evaluating the practical performance of this critical routine. In practice, its running time is shown to be constant with respect of the total number of operations: it is *100 times faster than the full application of the greedy algorithm* (that is, considering that all operations are impacted, implying that $\bar{n} = n$) and *2000 times faster than exact algorithms* (tests conducted with the simplex algorithm of the linear programming library GLPK 4.24). On the other hand, the total volume delivered by the routine is close to the optimal assignment, in particular when no stockout appears (the average gap between the greedy assignment and an optimal one is lower than 2 %).

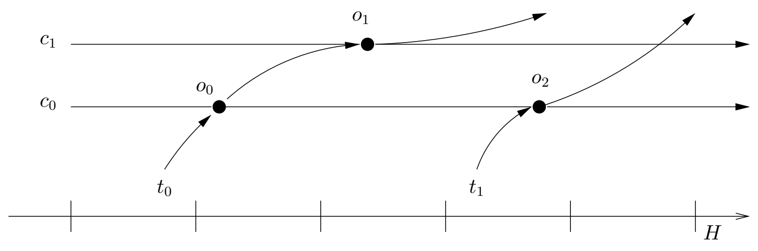

Figure 6 Bad configuration for the greedy volume assignment.

Finally, having assigned volumes, computing the gain of the transformation is done efficiently. The (variation of) cost of shifts is computed during the scheduling, and the (variation of) total delivered quantity is obtained during the assignment of volumes, without increasing the complexity of the algorithms. The (variation of) stockouts costs are also computed during the assignment of volumes. Note that computing the number of time steps in stockout between two consecutive operations requires $O(\log H)$ time, with H the number of time steps over the horizon, since it is equivalent to the problem of searching the zero of a discrete non-increasing function.

5.3.3. Implementation details. Although conceptually simple, the practical implementation of the evaluation routines is considerably complicated by incremental aspects. Even if such points are outside the scope of this paper, some important details of implementation are outlined below.

All sets (unordered or ordered, fixed or dynamic) are implemented as arrays, in order to improve the cache memory locality. Memory allocation is avoided as much as possible during local-search iterations: all the data structures are allocated before starting the local search; an array of capacity n representing a dynamic list is extended if necessary by reallocating a larger block of memory of size $n + k$ (with $k \approx 10$). The most frequently used data structure is the one for maintaining an unordered list of elements; although implemented as an array, it is designed to support basic routines (find, insert, delete, clear) in $O(1)$ time.

Since the success rate of transformations is low on average (a few percents), the rollback routine must be very efficient. In this way, the decision variables of the problem (for example, the starting and ending dates of an operation) are duplicated in such a way that only temporary data are modified by a transformation. In this way, the rollback routine consists simply in overwriting temporary data by current ones (that is, corresponding to the current solution). But this is complicated by the fact that during one transformation, several objects (in particular operations or shifts) are likely to move in the arrays in which they are stored. In order to ensure the (temporary) insertion or deletion of one object in $O(1)$ time, the objects of the array are doubly linked (see Figure 7 illustrating the exchange of operations between shifts).

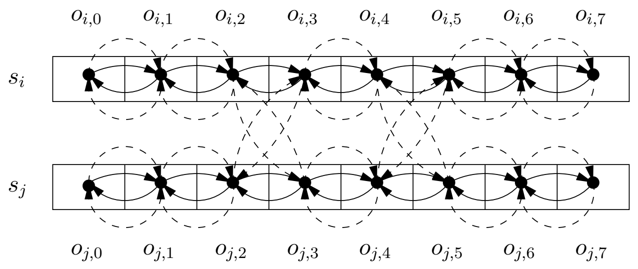

Figure 7 Representation of shifts and operations.
Note. Operations $o_{i,3}, o_{i,4}$ of the shift s_i are swapped with operations $o_{j,3}, o_{j,4}$ of the shift s_j: the current links (before transformation) are plain, the temporary links (after transformation) are dashed.

Finally, we focus on a data structure which is particularly critical for efficiency. Operations or shifts correspond to intervals over the horizon, for which we have the following need: given a date over the horizon, find the previous, current, or next operation performed at a site if any. The same problem arises for situating shifts performed by a resource. For this, the following data structure is employed. Assume that the n operations are stored into an ordered list L. The horizon is divided into m intervals $U_0, \dots, U_{m-1}$ of given length u (with u dividing T). Then, an array I is defined such that I_i refers to the first operation whose starting date is larger than the left endpoint of U_i. The next operation after date d is found by searching the operations between the one pointed by I_i with $i = \lfloor d/u \rfloor$ and the one pointed by I_{i+1}. In this way, the search is done in $O(k)$ time in the worst case, with k the number of operations contained in the interval U_i. If u corresponds to the entire horizon ($m = 1$), then $k = n$; on the other hand, if u corresponds to the smallest granularity for expressing time (here the minute, leading to $m = 21600$), then $k = 1$. Assuming that starting dates of operations performed at a site are uniformly distributed over the horizon, the number k is equal to n/m. In this case, searching takes $O(n/m)$ time (when evaluating the transformation), but the array I requires $O(m)$ space to be stored and $O(m)$ time to be updated (when committing the transformation). This implies two compromises: time to evaluate vs. time to commit, time to evaluate vs. space.

Theoretically, the best value m^* for solving the compromise on running time corresponds to the minimum of the function $T(m) = E(N/m) + Cm$, with N the average number of operations per customer and E (resp. C) a coefficient relative to the proportion of calls of the evaluate routine (resp. commit routine) per customer. A simple calculation using differentiation yields $m^* = \sqrt{(EN)/C}$. Table 2 shows the values of m^* for different realistic configurations of parameters N, E, C. In practice, we have chosen $m^* = 15$, which corresponds to interval U_i of one day: such a value of m offers a good compromise for running time (even in worst-case situations) and leads to a small memory footprint.

Table 2 Theoretical values of m^*.

N	2	2	2	10	10	10	30	30	30
E	90	95	99	90	95	99	90	95	99
C	10	5	1	10	5	1	10	5	1
m^*	4.2	6.2	14.1	9.5	13.8	31.5	16.4	23.9	54.5

6. Computational experiments

The whole algorithm was implemented in C# 2.0 programming language (for running on Microsoft .NET 2.0 framework). The resulting program includes nearly 30 000 lines of code, whose 6 000 lines (20 %) are dedicated to check the validity of all incremental data structures at each iteration (only active in debug mode). The whole project (specifications, implementation, tests), realized during the year 2008, required nearly 300 man-days. All statistics and results presented here have been obtained (without parallelization) on a computer equipped with a Windows Vista operating system and a chipset Intel Xeon X5365 (CPU 3 GHz, L1 cache 64 Kio, L2 cache 4 Mio, RAM 8 Go). The interested reader is invited to contact the authors to obtain some benchmarks to work on this problem. Note that the urgency-based constructive heuristic used to compute an initial solution is also called "greedy algorithm" below.

Since the local-search heuristic is randomized, 5 runs have been performed with different seeds for each benchmark. All results are presented in terms of cost (or cost per kilo) but it shall be noted that more than 80 % of the cost is proportional to the traveled distance, which means that our algorithm is likely to obtain similar gains in absence of fix costs (with a miles per ton objective). Except contrary mention, all the statistics presented below correspond to average results obtained for these 5 runs. Note that results requiring particular explanations are marked with asterisks (*) in figures presenting numerical experiments; these explanations could be found in the text below.

6.1. Short-term benchmarks

The local-search algorithm has been extensively tested on short-term benchmarks (15 days) with different characteristics: *realistic* (that is, matching the operational conditions), *pathological* (for example, with plants whose production is stopped several days), *very large-scale* (for example, with 1 500 sites and 300 resources). Some aggregated results are presented here for 17 short-term benchmarks involving from 46 to 500 customers. Table 3 gives the characteristics of these instances: the number of customers, plants, depots, drivers, tractors, and trailers. For these instances, our greedy algorithm finds a solution without stockout. Note that it is not always the case in operations (see results on long-term benchmarks).

The gains obtained by the local-search heuristic are presented in Table 4 using the results of the greedy algorithm as a reference. Two kinds of results are presented on this table: the gains obtained by optimizing directly the logistic ratio LR (denoted by LS-LR) and the ones obtained by optimizing the surrogate ratio LR' (denoted by LS-LR'). In both cases, the local-search heuristic improves drastically the quality of solutions provided by the greedy algorithm. The average gain obtained by LS-LR (resp. LS-LR') is of 29.2 % (resp. 45.4 %). Table 5 gives more statistics on the solutions found by local search. The column "nb shift" (resp. "nb oper") of the table reports the number of shifts (resp. operations) of the solution. The column "avg oper" (resp. "avg deliv", "avg load", "avg layov") reports the average number of operations (resp. deliveries, loadings, layovers) per shift. Finally, the column "avg dur" (resp. "avg dist") reports the average traveled distance (resp. duration) per shift. One can observe that more shifts and much more operations are included in both LS-LR and LS-LR' solutions, compared to ones produced by the greedy. On average, the number of shifts (resp. operations) is increased by nearly 25 % (resp. 50 %). In this way, the average number of operations per shift is increased from almost 4 to more than 6. The average distance and duration of shifts are decreased slightly in the case of LS-LR, whereas these ones are increased slightly in the case of LS-LR'.

The greedy algorithm is running in a few seconds even for large-scale instances (12 seconds for a test case with 1500 customers). Statistics about the performance of the local search are given on Table 6. The column "attempt" corresponds to the number of transformations attempted by the local-search heuristic. The columns "accept" (resp. "improve") corresponds to the number of accepted (resp. strictly improving) transformations; in addition, the corresponding rate for 100 (resp. 10 000) attempted transformations is specified. The local-search algorithm attempts more than 10 000 transformations per second, even for large-scale instances. On average, *our algorithm visits more than 10 million solutions in the search space during 5 minutes of running time* (which is the desired time limit in operational conditions). When planning over a 15-days horizon, the memory allocated by the program does not exceed 30 Mo for medium-size instances (hundred sites, ten resources), and 300 Mo for large-scale instances (thousand sites, hundred resources). The acceptance rate, which corresponds to the number of accepted transformations over the number of attempted ones, varies essentially between 1 and 10 %, with an average value close to 5 % over all the instances. Note that this rate is nearly constant all along the search, allowing a large diversification (without the use of metaheuristics). On the other hand, the number of strictly improving transformations is of several hundreds. One can observe that the choice of objective (LR or LR') does not affect the performance of the local-search heuristic.

6.2. Long-term benchmarks

The local-search algorithm has been tested on long-term benchmarks, in particular for verifying that optimizing the surrogate objective leads to better solutions in the long run. Some results are presented for 5 real-life benchmarks, each one with 105 days. The operational planning process is simulated as follows. The simulator starts at day 0 by computing a planning over the next 15 days, with 5 minutes as time limit. Then, only the shifts starting at the first day of this short-term planning are fixed (the levels of plants or customers visited by these shifts are updated, the resources operating on these shifts become unavailable) and the process is iterated the following day.

The characteristics of these 5 benchmarks are presented on Table 8. The complete statistics are given for each heuristic on Tables 9, 10, and 11. The main remark is that the average number of operations per shift is increased in local-search solutions. Indeed, the number of shifts in local-search solutions is slightly smaller than in greedy solutions, whereas the number of operations is increased. Besides, local-search solutions are characterized by a larger average traveled duration per shift, thanks to an increased use of layovers. The average logistic ratios marked by an asterisk are computed as the sum of shift costs for the 5 benchmarks divided by the sum of all delivered quantities. The gains obtained by LS-LR' are reported on the right part of Table 8. The column "wst 1 mn" reports the worst LR gain in % obtained over the 5 runs for 1 minute of running time per planning iteration. The column "avg 1 mn" (resp. "avg 5 mn", "avg 1 h") reports the average gain in % for LR obtained within 1 minute (resp. 5 minutes, 1 hour) of computation per planning iteration. Note that the solutions found by the greedy algorithm include some stockouts. On average, the LR gain obtained by LS-LR' with only 1 minute of running time per planning iteration is of nearly 20 % on average. These statistics demonstrate that our local-search heuristic, beyond delivering high-quality solutions, is fast and robust (with exponential-inverse convergence).

Solutions provided by logistic experts are reported on Table 13. Note that the comparison between the experts and the three heuristics is not completely fair. Indeed, because finding solutions with no stockout (with actual safety levels) was difficult and fastidious, experts were allowed to modify the initial long-term benchmarks in order to provide solutions without stockout. This could explain the negative gain of greedy algorithm on instances L1, L2, L4. Moreover, the solution provided by experts for instance L2 is considered as a "best-effort" solution, in the sense that many more time has been spent to optimize carefully the solution.

6.3. Impact of the surrogate objective

The key figures for comparing LS-LR and LS-LR' are given on Table 7 (for short-term benchmarks) and on the right part of Table 12 (for long-term benchmarks). "avg DQ" corresponds to the average total delivered quantity, and "avg delivq" to the average delivered quantity (per operation). On short-term benchmarks, the total delivered quantity in both LS-LR and LS-LR' solutions is increased by more than 50 % on average. But note that the average quantity per delivery is increased by almost 5 % in LS-LR' solutions compared to LS-LR solutions.

On long-term benchmarks, one shall observe that LS-LR' aims at increasing the average quantity per delivery, which results in better long-term solutions. On instances L1 (resp. L2), the augmentation is of 21 % (resp. 13 %), leading to logistic ratio savings of 10 % (resp. 7 %) compared to LS-LR. Moreover, LS-LR' is able to produce solutions without stockout on the instance L4, contrary to LS-LR. The values marked by an asterisk on Tables 12 and 13 are computed globally, for the 5 benchmarks (as done for logistic ratios on Tables 10 and 11).

7. Conclusion

Three contributions have been presented in this paper. First, a real-life IRP model encountered in a worldwide industry has been introduced. Then, a surrogate objective based on local lower bounds was defined for ensuring a long-term optimization when building a planning over the short term. Our experiments show that using this objective decreases the final logistic cost by 4 % on average, compared to a direct minimization of the logistic ratio. Finally, a local-search heuristic has been described for solving effectively and efficiently the real-life IRP over the short term (15 days in full details), even when some large-scale instances (thousand sites, hundred resources) are considered. By assigning volumes approximately but thousands times faster than with a classical flow algorithm, we are able to handle the problem as a whole, without decomposition. Using a large large variety of randomized neighborhoods, our first-improvement descent visits millions of solutions during the allocated execution time. Within 5 minutes of running time, we obtain long-term savings of 20 % on average compared to urgency-based strategies used by expert planners or implemented in classical insertion heuristics. The resulting decision support system is progressively deployed worldwide, confirming the promised gains in the field.

New researches are still conducted in several directions: further enlarging the scope of the IRP addressed in the paper (in particular, refining costs and managing drivers' desiderata); improving the existing local-search heuristic (adding transformations with larger neighborhoods); reinforcing the global lower bound by integrating tours visiting several sites and constraints on resources. Another prospective but promising line of research is to proceed step by step toward a global optimization of the supply chain, by tackling jointly the production and distribution problems, and ultimately by integrating purchasing issues. Indeed, we think that local-search approaches like the one developed presently are best suited for solving such very large-scale problems, while inducing new algorithmic challenges.

Acknowledgments

First, we warmly thank M. Saul Pedraza Morales (École de Mines de Nantes), which worked on several topics in relation with this paper and performed some of the numerical experiments (in particular about the practical efficiency of our volume assignment algorithm), during its internship at Bouygues e-lab in 2009. Then, we thank Dr. Philippe Baptiste (LIX, École Polytechnique) for suggesting to model inventory constraints as a flow network. We also express our gratitude to the two anonymous referees and the associate editor for their reviewing, which has resulted in an improved paper.

References

Aarts, E., J. Lenstra, eds. 1997. *Local Search in Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, Chichester, England.

Archetti, C., M. Savelsbergh. 2007. The truckload trip scheduling problem. *TRISTAN VI, the 6th Triennial Symposium on Transportation Analysis*. Phuket Island, Thailand.

Bard, J.F., F. Huang, M. Dror, P. Jaillet. 1998. A branch and cut algorithm for the VRP with satellite facilities. *IIE Transactions* **30**(9) 831–834.

Bell, W., L. Dalberto, M. Fisher, A. Greenfield, R. Jaikumar, P. Kedia, R. Mack, P. Prutzman. 1983. Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer. *Interfaces* **13**(6) 4–23.

Benoist, T., B. Estellon, F. Gardi, A. Jeanjean. 2009. High-performance local search for solving inventory routing problems. H. Hoos T. Stützle, M. Birattari, ed., *SLS 2009, the 2nd International Workshop on Engineering Stochastic Local Search Algorithms*, *Lecture Notes in Computer Science*, vol. 5752. Springer, 105–109.

Campbell, A., L. Clarke, A. Kleywegt, M. Savelsbergh. 1998. The inventory routing problem. T. Crainic, G. Laporte, eds., *Fleet Management and Logistics*. Kluwer Academic Publishers, Norwell, MA, 95–113.

Campbell, A., L. Clarke, M. Savelsbergh. 2002. Inventory routing in practice. P. Toth, D. Viego, eds., *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications 9, SIAM, Philadelphia, PA, 309–330.

Campbell, A., M. Savelsbergh. 2004a. A decomposition approach for the inventory-routing problem. *Transportation Sci.* **38**(4) 488–502.

Campbell, A., M. Savelsbergh. 2004b. Delivery volume optimization. *Transportation Sci.* **38**(2) 210–223.

Campbell, A., M. Savelsbergh. 2004c. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Sci.* **38**(3) 369–378.

Cormen, T., C. Leiserson, R. Rivest, C. Stein. 2004. *Introduction à l'Algorithmique*. Dunod, Paris, France. French 2nd edition.

Dror, M., M. Ball. 1987. Inventory/routing: reduction from an annual to a short-period problem. *Naval Research Logistics* **34**(6) 891–905.

Estellon, B., F. Gardi, K. Nouioua. 2006. Large neighborhood improvements for solving car sequencing problems. *RAIRO Operations Research* **40**(4) 355–379.

Estellon, B., F. Gardi, K. Nouioua. 2008. Two local search approaches for solving real-life car sequencing problems. *Eur. J. Oper. Res.* **191**(3) 928–944.

Estellon, B., F. Gardi, K. Nouioua. 2009. High-performance local search for task scheduling with human resource allocation. H. Hoos T. Stützle, M. Birattari, ed., *SLS 2009, the 2nd International Workshop on Engineering Stochastic Local Search Algorithms*, *Lecture Notes in Computer Science*, vol. 5752. Springer, 1–15.

Feo, T., G. Resende. 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization* **6**(2) 109–133.

Lau, H., Q. Liu, H. Ono. 2002. Integrating local search and network flow to solve the inventory routing problem. *AAAI 2002, the 18th National Conference on Artificial Intelligence*. AAAI Press, Menlo Park, CA, 9–14.

Savelsbergh, M., J.-H. Song. 2007a. Inventory routing with continuous moves. *Computers and Operations Research* **34**(6) 1744–1763.

Savelsbergh, M., J.-H. Song. 2007b. Performance measurement for inventory routing. *Transportation Sci.* **41**(1) 44–54.

Savelsbergh, M., J.-H. Song. 2008. An optimization algorithm for the inventory routing with continuous moves. *Computers and Operations Research* **35**(7) 2266–2282.

Solomon, M. 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.* **35**(2) 254–265.

Appendix

Table 3 Short-term benchmarks characteristics (17 test cases).

	customers	plants	depots	drivers	tractors	trailers
minimum	46	1	1	10	5	10
maximum	500	5	2	50	50	50
average	138	1.9	1.2	28	19	25

Table 4 Short-term benchmarks: local search results.

	LS-LR	LS-LR'
minimum gain over greedy	20.8 %	28.5 %
maximum gain over greedy	39.1 %	65.6 %
average gain over greedy	29.2 %	45.4 %

Table 5 Short-term benchmarks: statistics on shifts (average over all test cases).

algorithm	nb shift	nb oper	avg oper	avg deliv	avg load	avg layov	avg dist	avg dur
greedy	68.6	473.6	4.9	2.8	2.2	0.2	481.8	676.8
LS-LR	91.8	689.6	5.9	3.3	2.6	0.2	399.7	617.5
LS-LR'	86.7	692.2	6.4	3.5	2.9	0.2	487.9	708.4

	attempt	accept		improve	
minimum	4.867 M	141 962	2.6 %	497	0.4 h%
maximum	16.167 M	705 213	6.6 %	2 067	4.2 h%
average	8.596 M	377 169	4.5 %	937	1.3 h%

	attempt	accept		improve	
minimum	3.706 M	117 333	2.4 %	389	0.5 h%
maximum	15.184 M	536 327	6.8 %	2 218	5.2 h%
average	7.830 M	303 620	4.1 %	946	1.5 h%

Table 6 Short-term benchmarks: statistics on transformations for LS-LR (left) and LS-LR' (right) optimization. M = million, h% = one-hundredths percent.

Table 7 Short-term benchmarks: statistics on volumes.

avg DQ greedy	avg DQ LS-LR	avg DQ LS-LR'	avg delivq greedy	avg delivq LS-LR	avg delivq LS-LR'
3 031 400	4 565 518	4 861 465	15 992	16 066	17 073

Table 8 Long-term benchmarks: characteristics and LR gains with different time limits.

data	customers	plants	depots	drivers	tractors	trailers	callins	orders	wst 1 mn	avg 1 mn	avg 5 mn	avg 1 h
L1	75	6	1	35	21	5	19	56	23.8 %	24.6 %	26.3 %	26.5 %
L2	75	6	1	35	21	5	20	55	22.3 %	23.5 %	24.9 %	25.2 %
L3	175	8	1	35	21	12	36	189	5.2 %	5.8 %	8.3 %	11.2 %
L4	165	4	1	24	11	7	33	167	9.9 %	11.2 %	14.0 %	18.9 %
L5	198	8	7	12	12	12	3	40	32.5 %	34.2 %	35.7 %	35.9 %
average	138	6	2	28	17	8	22	101	18.7 %	19.9 %	21.8 %	23.5 %

Table 9 Long-term benchmarks: greedy results.

data	*SO*	*SC*	*DQ*	*LR*	nb shift	nb oper	avg oper	avg deliv	avg load	avg layov	avg dist	avg dur
L1	652	406 443	3 767 868	0.107 871	189	503	2.7	1.6	1.0	0.6	640	1 320
L2	146	407 379	3 827 560	0.106 433	196	506	2.6	1.6	1.0	0.5	619	1 235
L3	86	1 092 976	31 989 357	0.034 167	790	3 584	4.5	2.7	1.8	0.2	366	954
L4	257	808 887	18 433 289	0.043 882	590	2 249	3.8	2.4	1.4	0.2	395	844
L5	85	145 339	8 830 708	0.016 458	295	1 020	3.5	1.9	1.5	1.2	598	1 760
average	245	572 205	13 369 756	*0.042 798	412	1 572	3.4	2.0	1.3	0.5	524	1 223

Table 10 Long-term benchmarks: LS-LR results.

data	*SO*	*SC*	*DQ*	*LR*	nb shift	nb oper	avg oper	avg deliv	avg load	avg layov	avg dist	avg dur
L1	0	340 767	3 840 502	0.088 730	137	590	4.3	3.0	1.3	0.8	721	1 618
L2	0	335 661	3 899 780	0.086 072	148	570	3.9	2.7	1.2	0.7	660	1 445
L3	0	1 019 292	32 079 238	0.031 774	839	3 570	4.3	2.6	1.7	0.2	317	873
L4	17	697 009	18 694 845	0.037 283	605	2 400	4.0	2.6	1.4	0.2	321	735
L5	0	106 326	9 475 562	0.011 221	110	1 324	12.0	7.8	4.3	3.2	1 256	4 286
average	3	499 811	13 597 985	*0.036 756	368	1 691	5.7	3.7	2.0	1.0	655	1 792

Table 11 Long-term benchmarks: LS-LR' results.

data	*SO*	*SC*	*DQ*	*LR*	nb shift	nb oper	avg oper	avg deliv	avg load	avg layov	avg dist	avg dur
L1	0	321 449	4 045 989	0.079 449	148	542	3.7	2.4	1.3	0.7	632	1 403
L2	0	321 207	4 016 621	0.079 969	140	541	3.9	2.6	1.3	0.7	669	1 471
L3	0	1 012 191	32 320 180	0.031 318	807	3 583	4.4	2.7	1.8	0.2	327	890
L4	0	701 139	18 587 949	0.037 720	602	2 396	4.0	2.6	1.4	0.2	325	744
L5	0	101 913	9 630 979	0.010 582	138	1 352	9.8	6.3	3.5	2.2	945	3 159
average	0	491 580	13 720 344	*0.035 829	367	1 683	5.2	3.3	1.9	0.8	580	1 533

Table 12 Long-term benchmarks: gains of LS-LR' against LS-LR (left) and gains obtained by local search against greedy (right).

data	LS-LR' against LS-LR
L1	10 %
L2	7 %
L3	1 %
L4	*-1 %
L5	6 %

data	gain LS-LR	gain LS-LR'	avg delivq greedy	avg delivq LS-LR	avg delivq LS-LR'
L1	17.7 %	26.3 %	12 460	9 344	11 391
L2	19.1 %	24.9 %	12 205	9 759	11 035
L3	7.0 %	8.3 %	14 997	14 706	14 833
L4	15.0 %	14.0 %	13 018	11 885	11 876
L5	31.8 %	35.7 %	15 755	11 044	11 078
average	*14.1 %	*16.3 %	*14 146	*12 537	*12 864

Table 13 Long-term benchmarks: gains obtained by local search against logistic experts.

data	*SO*	*SC*	*DQ*	*LR*	gain greedy	gain LS-LR	gain LS-LR'
L1	0	378 778	3 725 847	0.101 662	-6.1 %	12.7 %	21.9 %
L2	0	328 364	3 567 370	0.092 047	-15.6 %	6.5 %	13.1 %
L3	0	1 257 354	32 667 576	0.038 489	11.2 %	17.4 %	18.6 %
L4	0	788 893	18 683 473	0.042 224	-3.9 %	11.7 %	10.7 %
L5	0	290 921	10 398 050	0.027 978	41.2 %	59.9 %	62.2 %
average	0	608 862	13 808 463	*0.044 093	*2.9 %	*16.6 %	*18.7 %