Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 195 (2009) 703–715
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

A memetic algorithm with dynamic population management for an integrated production–distribution problem

M. Boudia, C. Prins *

Institute Charles Delaunay, University of Technology of Troyes, BP 2060, 10010 Troyes, France

Received 11 January 2007; accepted 17 July 2007
Available online 13 November 2007

Abstract

This paper studies an NP-hard multi-period production–distribution problem to minimize the sum of three costs: production setups, inventories and distribution. This problem is solved by a very recent form of metaheuristic called memetic algorithm with population management (MA|PM). Contrary to classical two-phase methods (production planning, then distribution planning), the algorithm simultaneously tackles production and distribution decisions. Several versions with different population management strategies are evaluated and compared with a two-phase heuristic and a Greedy Randomized Adaptive Search Procedure (GRASP), on 90 randomly generated instances with 20 periods and 50, 100 or 200 customers. The significant savings obtained compared to the two other methods confirm both the interest of integrating production and distribution decisions and of using the MA|PM template.

Keywords: Production; Distribution; Inventory routing; Memetic algorithm

1. Introduction and problem statement

This paper considers an integrated production–distribution problem or IPDP, defined on a complete weighted and undirected network $G = (V, E, C)$. V is a set of $n + 1$ nodes indexed from 0 onwards. Node 0 represents a plant with a limited fleet of f identical vehicles of limited capacity W, while the other n nodes denote customers. $E = \{(i, j) : i \in V, j \in V, i \neq j\}$ is the edge set and the weight $C_{ij} = C_{ji}$ on each edge (i,j) is the travelling cost between nodes i and j.

The plant manufactures some products to supply the customers over a planning horizon T subdivided into $m + 1$ periods or "days", indexed from 0 to m. Index 0 denotes a fictitious period used for initial inventories. Like in vehicle routing literature, it is assumed that the products do not need to be distinguished, like bags or pallets: they do not create resource conflicts in production (dedicated production lines for instance) and they can share the same storage areas or vehicles. Hence, in the sequel, the amounts of the different products are aggregated into one single equivalent product.

The plant has a limited production capacity per day PPC. If production is decided in a given day t, one fixed setup cost s is incurred. The product may be stored at the plant, with a limited storage capacity PSC and a storage cost h per unit of product and per day. It may be also distributed to customers using the fleet of vehicles. Each customer i has a known demand $q_{it} \leqslant W$ for each day t and a limited storage area of capacity CSC_i. Inventory holding costs at customers are supported by them and therefore ignored in our problem.

Shortages are not allowed. Customers can be serviced in advance, provided their storage capacities are respected, but not late. Each customer can be serviced at most once a day and each vehicle is limited to one trip per day. It is forbidden to split the demands, i.e., each q_{it} must be supplied by one single trip. The deliveries to each customer i are submitted to a FIFO policy: q_{it} must not be delivered

* Corresponding author.
E-mail addresses: boudia@utt.fr (M. Boudia), prins@utt.fr (C. Prins).

0377-2217/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2007.07.034

later than q_{iu} if $t \leqslant u$. Several customers may be visited by the same trip (less-than-truckload distribution). The cost of a trip is the total cost of traversed edges and the distribution cost in a day is the sum of all trip costs for this day.

By convention, and without loss of generality, deliveries take place in the morning, before a production run at the plant and before the customers' daily consumption. The production of each day is available the next morning. This kind of convention is common in production planning models, to write inventory balance equations and to know when storages costs must be counted. In our case, its goal is also to impose an order between production and distribution activities in each period, to avoid for instance scheduling problems if vehicles have to wait for the product or vice versa.

The objective is to determine, for each period, the amount produced at the plant, the quantities shipped to each customer and the associated trip sequences, to minimize total cost (setup costs, holding costs and distribution costs). The problem is extremely hard: the periodic vehicle routing problem (PVRP), known to be NP-hard [11], is the particular case obtained if the plant has an infinite production capacity and if setup and storage costs are ignored. Note that customers place no orders: their stocks are directly managed by the plant and the IPDP can be viewed as an inventory routing problem, combined with a production planning problem.

The IPDP is a step towards more realism in logistic research, since it integrates a production planning problem and a vehicle routing problem. Of course, it can be complicated in practice by various additional constraints, e.g., heterogeneous vehicles or several products. Moreover, the data require an efficient information system to track the demands of customers. Typically, like in production planning, the planning horizon covers a few weeks or months and the production and distribution plans are refreshed every week using the "rolling horizon" technique. Therefore, the computational time is less critical than in shop scheduling for instance, in which a new schedule must be computed everyday. However, running times must remain reasonable (e.g., 1 h maximum), to enable the decision maker to evaluate a few scenarios and make simulations.

The main topic of this paper is a solution method able to simultaneously tackle production and distribution decisions. More precisely, the method is based on a very recent kind of metaheuristic, the memetic algorithm with population management or MA|PM. The paper is organized as follows: Section 2 underlines problem relevance and presents related works in literature. The MA|PM and its components are described in Section 3, while Section 4 is devoted to computational evaluations. A conclusion and some future directions are presented in Section 5.

2. Relevance and literature

Production planning and vehicle routing are two complex disciplines, explaining why most companies adopt a two-phase approach, in which vehicle trips are elaborated after the production plan. The same partition can be observed in research. Production planning and distribution planning have been widely but separately investigated.

On one hand, distribution problems like the VRP (vehicle routing problem) represent an active research field, see for instance the book by Toth and Vigo [38]. The best exact method for the VRP is a branch-and-cut-and-price developed by Fukasawa et al. [16]. Cordeau et al. [13] survey recent metaheuristics for large instances, like the memetic algorithm of Prins [29]. Extensions like stochastic cases [18], inventory constraints [10,14] or multi-period horizons [19,11] have been studied. However, production decisions are usually ignored or only partially handled.

On the other hand, production planning has been studied in detail but without taking distribution into account [21,27]. Recent surveys include a book chapter by Graves [20] and a book by Voss and Woodruff [40]. Integer linear models seem well adapted to production planning, see for instance Chen and Ji [9]. It is true that supply chain management research tries to design more global approaches that encompass both suppliers and end-customers, but it mainly concentrates on qualitative or strategic issues like supplier and market selection or outsourcing [5].

A review of the different problems raised by coordination between production and distribution can be found in Sarmiento and Nagi [32] and in the Ph.D. thesis of Turano [39]. It appears that very few papers consider combinatorial optimization problems raised by the integration of production and distribution at the tactical and operational levels, like our IPDP. However, such problems are important, especially when distribution costs are of the same order as production costs, e.g. mineral waters, concrete or livestock feed.

The IPDP as defined here was introduced in one of our earlier works [2]. An integer linear programming model was proposed. A two-phase or uncoupled heuristic called H1 was developed to reflect common practice in industry. In short, the first phase of H1 elaborates a production plan using Wagner and Whitin's dynamic programming (DP) method, see for instance [6]. This DP method was adapted to the limited production capacity in the IPDP, without loosing its polynomial complexity. In the second phase, this plan is frozen, the trips are built for each day and a local search is applied day by day to improve them. A three-phase heuristic H2 was also designed: its two first phases correspond to H1, while the last phase is a kind of feedback on the production plan which determines definitive production dates for the quantities to be delivered. H2 was compared to H1 and brought significant savings, for instance 13% on average for randomly generated instances with 200 customers and 20 periods.

More recently, we have studied a Greedy Randomized Adaptive Search Procedure or GRASP [3]. The initial solution at each iteration of the GRASP is built with a randomized greedy constructive heuristic, taking production and distribution levels into account. This solution is then

improved using a more powerful local search than in H1 and H2: the moves considered can remove a customer from a trip and reinsert him in a trip belonging to another day. The production plan is modified accordingly. Compared to H1, this GRASP, which can be considered as a true integrated approach, provides solution values reduced by 17% on the instances already mentioned.

Chandra and Fisher [8] studied a similar problem, but some differences must be underlined. They consider several products, an unlimited fleet size and split deliveries. The plant has an unlimited storage capacity and inventory holding costs are not considered. Compared to our case, the multi-product option seems more complicated, but in fact the unlimited fleet and split deliveries make the problem easier, because the hard bin packing sub-problems consisting of assigning the demands to a limited number of vehicles are avoided. An integrated approach is proposed and applied to smaller instances with up to 10 products, 50 customers and 10 periods. Savings between 3% and 20% are reported, compared to a two-phase approach. It should be noted that their instances are weakly constrained: one third of them have a total demand per day limited to 85% of production capacity, one other third 60%, while the last third considers an unlimited production capacity.

Chandra [7] described a problem of preparation of orders in a warehouse to satisfy the demands of customers in the same region. If an order cannot be satisfied, it may be transmitted to a higher echelon like a factory, but this induces a fixed cost. The tests on 33 different data sets show a cost reduction ranging from 5% to 14% when distribution and order preparation are coordinated. Fumero and Vercellis [17] dealt with a problem almost identical to the one studied by Chandra and Fisher. Their solution method, based on Lagrangean relaxation, was tested on even smaller instances, with up to 12 customers, 10 products and 8 periods. Here again, the algorithm was compared with an uncoupled approach and significant gains were obtained.

Other problems of coordination between production and distribution in a broader sense can be found in postal activities, e.g., Metters [25] investigated the coordination between a sorting centre and mail distribution. The objective not only includes the total cost, but also the reduction of routing delays. Bramel et al. [4] treated a problem of cooperation between several factories that make components or sub-assemblies for each other. However, the routing aspects are discarded, since truckload transportation is assumed between factories.

3. MA|PM components

3.1. Principles of MA|PM

In combinatorial optimization, classical genetic algorithms (GA) are not aggressive enough compared to other metaheuristics like tabu search. Memetic algorithms (MA) are more powerful versions proposed by Moscato [26], in which intensification is brought by an improvement procedure (local search) applied to each initial solution and to the offspring obtained by crossover and mutation. For some classical optimization problems, memetic algorithms are currently the best solution methods. For instance, very efficient MAs have been designed by Prins [29] for the VRP and by Lacomme et al. [23] for the CARP (capacitated arc routing problem). The CARP is a routing problem raised by municipal refuse collection, in which a set of arcs (streets) must be serviced by vehicles.

Many researchers agree that the quality of a metaheuristic is largely the result of the interplay between intensification and diversification, see Ferland et al. [15] and Laguna et al. [24]. As Hertz and Widmer pointed out [22], preserving population diversity is essential in evolutionary algorithms. This problem is even more critical in memetic algorithms, because the local search tends to concentrate solutions in a few attraction basins. This can be countered by increasing mutation rate, but then convergence is much slower. Recently, Tang et al. [37,36] have solved this problem in parallel generational memetic algorithms. They apply the local search selectively, using a selection ratio based on the right half of a Gaussian distribution. This ratio is designed to progressively decrease the number of selected solutions, as the number of generations increases.

The MA with population management or MA|PM is another recent way of tackling the diversity problem, introduced by Sörensen in his Ph.D. thesis [33] and recently published in a journal [34]. Like in any other incremental (steady state) MA, starting from an initial population, each iteration consists of selecting two parents, applying a crossover operator, and replacing some existing solutions by the offspring. However, MA|PM works on smaller populations (20–30 solutions) and mutation is replaced by a diversity control based on a distance measure in solution space.

More precisely, for a population P, let $d(B,C)$ be the distance between two solutions B and C of P and, by extension, $D_P(C) = \min\{d(B,C) : B \in P\}$ the distance of a new solution C to P. C is accepted to replace one solution in P if and only if $D_P(C) \geqslant \Delta$, where Δ is a given diversity threshold that can be dynamically adjusted during the search. In other words, new solutions are accepted only if they differ enough from existing solutions, in terms of structure. This diversification technique is powerful enough to avoid varying the local search rate like in [37,36].

In his original version, Sörensen recommends a systematic local search. Whenever a new solution does not pass the distance test ($D_P(C) < \Delta$), he suggests to apply a mutation operator until the child is accepted, but the resulting solution can be strongly degraded. Prins et al. tried to avoid this mutation operator by simply discarding the child. However, the local search rate must be reduced to avoid rejecting too many children. This version of MA|PM with local search rates between 20% and 50% has been successfully applied to two vehicle routing problems: the capacitated arc routing problem or CARP [31] and the location routing problem or LRP [30].

It is interesting to note that MA|PM is a kind of missing link between GAs and scatter search: both families of algorithms are population-based methods which apply local improvement and diversification techniques based on distance measures in solution space. However, according to our experience, MA|PM has a simpler structure and is easier to implement. For instance, a GA can be upgraded by adding a local search to give an MA, and then by adding a distance measure to obtain an MA|PM. Moreover, a dynamic control of diversity can be implemented in MA|PM, in lieu of the blind alternation between diversification and intensification in scatter search.

The following subsections describe the components used in our MA|PM for the IPDP. A pseudo-code is given in Algorithm 1, in Section 3.7.

3.2. Solution encoding and evaluation

Due to the complexity of our problem and for efficiency reasons, the MA|PM works directly on complete solutions instead of true chromosomes. A solution is completely defined by a tuple (T, X, Y, Z). Other solution attributes are easily deduced from this tuple, e.g., the plant inventory in day t, PI_t, or the inventory of each customer i in each day t, CI_{it}.

T is a list containing the trips of successive periods. Each trip is a sequence of distinct customer indexes (integers between 1 and n). Two consecutive trips in the same day are separated by one copy of the depot node (node 0). This trip delimiter is replaced by a day delimiter (-1) between two trips pertaining to two consecutive days. Days without deliveries can be recognized in T by two consecutive day delimiters. T has a varying length because the number of trips and the number of visits to each customer depend on solutions.

X is an $n \times m$ matrix in which X_{it} denotes the amount brought to each customer i in each period $t \neq 0$. Recall that day 0 is a fictitious day without distribution, used for initial inventories. X_{it} may include several demands q_{iu}, $u \geqslant t$, which can be easily deduced thanks to the FIFO delivery policy. Using T, X and the initial inventories CI_{i0} of customers, it is also possible to compute the inventory levels of customers (CI_{it}) and to check that the capacity W of vehicles, the fleet size f and all customer storage capacities CSC_i are respected. An integer linear program linking all these variables can be found in our GRASP paper [3].

Y is an $n \times (m+1)$ matrix, in which $Y_{it}, t \geqslant 0$, represents the production day of demand q_{it}. Day 0 means that the demand is satisfied with the initial plant inventory, PI_0. The amount PI_t stored at the plant in each period t can be derived from Y, X and PI_0 to check the plant storage capacity PSC. The last field Z is an m-vector in which Z_t denotes the total amount produced in day t. Z is redundant since it can be calculated from Y: however, we prefer to individualize it because it plays a role in the distance computation (see 3.6).

The cost of the solution associated with one tuple comprises a production term and a distribution term. Knowing that each trip begins and end at the depot, the distribution cost or total cost of the trips can be obtained from T, by summing the travelling costs C_{ij} between adjacent nodes i and j. The production cost is obtained by multiplying the amounts stored each day by h and by counting one setup cost s for each day t such that $Z_t \neq 0$. More precisely, the cost of one solution can be computed using Eq. 1. In this equation, it is assumed that the Boolean expression $Z_t \neq 0$ returns 0 or 1 and that the day delimiter -1 behaves like the depot in terms of travelling costs, i.e., $C_{i,-1} = C_{i0}$ for any node i:

$$\sum_{t=0}^{m-1} h \times PI_t + \sum_{t=1}^{m} s \times (Z_t \neq 0) + \sum_{i=1}^{|T|-1} C(T_i, T_{i+1}). \quad (1)$$

Fig. 1 depicts one small instance with two periods and the encoding of one feasible solution. There are two trips with loads 15 and 13 in day 1 and one trip with load 11 in day 2: fleet size and vehicle capacities are respected. Customers 1, 4 and 5 receive their total demand the first day, while customers 2 and 3 are supplied each day. All demands are satisfied without shortages. The amount shipped to customers in day 1 (28) comes from the initial plant inventory (31). There is one single production run, in day 1 (8 units). The amount produced plus the residual plant inventory are delivered in day 2. Production capacity and plant storage capacities are respected.

Assume a total cost equal to 100 for the trips, a setup cost $s = 50$ and a storage cost $h = 5$ per unit and per period at the plant. There is one production day only. The initial plant inventory is stored one day, the 3 units for client 3 one day more, and the 8 units for client 2 one day (they are delivered in day 2). The total cost of the solution is then $100 + (1 \times 50) + (31 \times 5) + (11 \times 5) = 360$.

3.3. Initial population

The initial population P contains nc randomly generated solutions. Each initial solution is constructed in three steps. Random decisions are taken in the first step only: the two other steps are deterministic. These three steps are detailed below.

Step 1. The total amount to be produced is equal to the total demand of customers over the horizon, minus the initial plant inventory PI_0. The minimum number of production days is then $\alpha = \lceil (\sum_{i=1}^{n} \sum_{t=1}^{m} q_{it} - PI_0)/PPC \rceil$. Randomly draw a number of production days k between α and $m-1$ (not m because production of day m could not be distributed). Then randomly select k distinct days in $[1, m-1]$. In each selected day, the amount required to cover customers consumption until the next production day is produced. Production capacity may be violated here, due to excessive spacing between production days. This will be repaired in step 3.

One instance with $m = 2$ days

Customer i:	1	2	3	4	5	$f = 2, W = 18, PPC = 10, PSC = 32.$
Demands for 1st day q_{i1}:	2	4	1	0	8	$CSC_i = 10$ for all i.
Demands for 2nd day q_{i2}:	5	8	3	7	1	$PI_0 = 31$, $CI_{i0} = 0$ for all i.

One feasible solution

Trip list T with customers:	0	4	1	3	0	2	5	-1	3	2	0
Associated delivery day t:		1	1	1		1	1		2	2	
Amount supplied X_{it}:		7	7	1		4	9		3	8	
Production day Y_{it}:		0	0	0		0	0		0	1	
Amounts produced Z_1, Z_2:		0	8								

Fig. 1. Example of instance and solution.

Step 2. Trip construction day per day. For each day t, we start by determining the amount X_{it} delivered to each customer i. This amount is initialized with the demands q_{it} of customers threatened by shortage in day t. Then, to take advantage of the free storage at customers, a maximum amount of demand is also satisfied in advance in day t. This is done by considering each day u from t to the next production day v and by testing each client i: q_{iu} is added to X_{it} if the storage capacity of i and the fleet capacity allow it. To avoid a hard bin packing problem with the limited fleet, the total amount distributed in each day t is provisionally restricted to a fraction β of fleet capacity, i.e. to βfW. We took $\beta = 0.9$ in all our experiments.

The trips are built using Clarke and Wright saving algorithm [12]. This algorithm starts from a trivial solution, with one dedicated trip per customer. Then, it performs successive mergers (concatenations of two trips) to reduce the total cost of the trips, until additional mergers would violate vehicle capacity or increase total cost. However, the fleet is limited in the IPDP and there can be more trips than vehicles at the end. The algorithm has been adapted to continue in that case with the least cost-increasing mergers, if any, because each merger saves one vehicle. The merging process stops when fleet size is respected (f trips).

Step 3. Let S_t be the total amount distributed in day t. This step adjusts the production day of each S_t to minimize the sum of setup costs and storage costs at the plant. This classical production planning problem can be solved using Wagner and Whitin's dynamic programming (DP) method [6], suitably modified to handle the limited production and storage capacities of the plant. The production capacity violations which can be created in step 1 are corrected here.

Concerning the complexity, the amounts distributed X_{it} are computed in $O(m^2n)$. The Clarke and Wright algorithm can be implemented in $O(n^2 \log n)$ for one period [28] and in $O(mn^2 \log n)$ for the whole horizon. The complexity of the DP method is $O(m^2)$. Hence, each initial solution can be generated in $O(m^2n + mn^2 \log n + m^2) = O(m^2(n+1) + mn^2 \log n) = O(m^2n + mn^2 \log n) = O(mn(m + n \log n))$.

3.4. Selection and crossover

In the sequel, $U \cdot V$ denotes the field V in the tuple representing a solution U, e.g., $A \cdot T$ is the list of trips for solution A. For a given trip list T, $T(i)$ denotes the element of rank i and $T(i \to j)$ the sublist containing the elements $T(i)$ to $T(j)$.

The two parents A and B for a crossover are chosen with the binary tournament method: two solutions are randomly drawn in P and the best one is kept for A, the same process is repeated to get B. A and B undergo a two-point crossover which returns one single child C. The two cutting points c_1 and c_2 must correspond to day or trip delimiters (0 or -1) in $A.T$, i.e. the crossover never splits a trip of A. To do this, let τ be the number of trips in A. Two distinct integers are randomly drawn between 1 and $\tau - 1$: 1 means a cut after the first trip while $\tau - 1$ means a cut after the last but one trip. The smallest number is assigned to c_1 and the largest to c_2.

Let d_1 and d_2 be the first and last days in $A \cdot T(c_1 \to c_2)$. $C \cdot T$ is composed of three sublists $C \cdot T_L$, $C \cdot T_M$ and $C \cdot T_R$. The left sublist $C \cdot T_L$ is initialized with the symbols corresponding to days 1 to $d_1 - 1$ in $B \cdot T$. The sublist $C \cdot T_M$ in the middle is initialized to $A \cdot L(c_1 \to c_2)$. The amounts $A \cdot X_{it}$ for the customers and days in $A \cdot L(c_1 \to c_2)$ are also copied in C. The right sublist $C \cdot T_R$ is initialized with the symbols of $B \cdot T$ for days $d_2 + 1$ to m. The list $C \cdot T$ is finally scanned to correct possible anomalies in the three sublists, while preserving as much as possible the central part inherited from A. Let i be the current symbol of $C \cdot T$ (delimiter or customer) reached by the scanning loop and t the associated day.

Correction of $C \cdot T_L$. If i is a day or trip delimiter, it is not modified. If i is a customer and if none of the demands delivered with $C \cdot X_{it}$ is already delivered in $C \cdot T_M$, i is not modified. Otherwise, the demands already satisfied in $C \cdot T_M$ are deducted from $C \cdot X_{it}$. If $C \cdot X_{it}$ becomes null, i is removed from his trip.

Correction of $C \cdot T_M$. The middle part of $C \cdot T$ is finalized by browsing the symbols for days d_1 to d_2 in $B \cdot T$. If the incumbent symbol i in $B \cdot T$ is a day or trip delimiter, it is ignored ($C \cdot T_M$ already contains such delimiters). If i is a customer, let U be the set of demands q_{iu} ($u \geqslant t$) delivered to i in day t of $B \cdot T$ but not delivered in C_M. Three cases are considered.

Case 1. U is empty. In that case, there is nothing to do for customer i.

Case 2. $U \neq \emptyset$ and customer i is visited in day t of $C \cdot T_M$. In that case, C_M is modified as follows:

(a) Add the demands of U to $C \cdot X_{it}$.
(b) If vehicle capacity is violated, i is removed from his current trip.
(c) A least-cost insertion of i is attempted in the existing trips of day t.
(d) If this insertion fails (no existing trip in day t offers enough capacity) and if C contains less than f trips, a new trip is created in day t to receive i.
(e) If all vehicles of the fleet are already used, we seek the trip with maximum residual capacity in day t and insert a maximum amount of demand into it. The remaining demands in U are sorted in chronological order. For each demand, the earliest day $u < t$ of C_L or C_M in which the demand can be inserted is determined, and a best insertion is performed in that day.

Case 3. $U \neq \emptyset$ and customer i is not already visited in day t of C_M. In that case, i is processed like in steps (c) to (e) of Case 2, i.e., a best insertion of i in the trips of day t or in earlier trips is performed.

Correction of $C \cdot T_R$. This sublist is corrected like $C \cdot T_L$, except that the $C \cdot X_{it}$ are here decremented by the demands already satisfied by $C \cdot T_L$ or $C \cdot T_M$ (instead of $C \cdot T_M$ only).

The correction process may remove some customers and create empty trips in C, delimited by -1 and 0 or by two zeros: such empty trips are eliminated by deleting one redundant zero. After this cleaning, child C is completed by taking the production day $C \cdot Y_{it}$ of each demand q_{it} from A for C_M and from B for C_L and C_R, provided the plant production and storage capacities are respected. Otherwise, a demand distributed on day t is produced the latest day $u < t$ compatible with these capacities. Finally, $C \cdot Z$ is deduced from $C \cdot Y$ and solution cost is computed.

Fig. 2 presents an example of crossover for two solutions A and B. The instance considered and solution A come from Fig. 1. The two cut points $c_1 = 4$ and $c_2 = 7$ define $A \cdot T(c_1 \to c_2) = (0, 2, 5, -1)$, shown in boldface, and $d_1 = d_2 = 1$. Hence, $C \cdot T_L$ is initialized to empty, $C \cdot T_M$ with $(0, 2, 5, -1)$ and $C \cdot T_R$ with $(2, 5)$. There is nothing to correct in the empty list $C \cdot T_L$.

$C \cdot T_M$ is corrected by scanning the symbols with day 1 in $B \cdot T$. Customer 1 falls in Case 3 because it is not already visited in day 1 of $C \cdot T_M$. Since he cannot be inserted in the existing trips of day 1, a new trip is created for him in $C \cdot T_M$. Then, customer 2 is ignored (Case 1) because his demands received in B are already satisfied in $C \cdot T_M$. Like customer 1, Case 3 applies to customer 3: it can be inserted in the new trip. The trip delimiter 0 cannot be copied because the two vehicles are already used in day 1. Then customer 4 falls in Case 3 and fills completely the new trip. Customer 5 corresponds to Case 1 and is ignored.

Parent A

Trip list T with customers:	4	1	3	**0**	**2**	**5**	**-1**	3	2
Associated delivery day t:	1	1	1		1	1		2	2
Amounts supplied X_{it}:	7	7	1		4	9		3	8
Production day Y_{it}:	0	0	0		0	0		0	1
Amounts produced Z_1, Z_2:	0	8							

Parent B

Trip list T with customers:	1	2	3	0	4	5	-1	2	5
Associated delivery day t:	1	1	1		1	1		2	2
Amounts supplied X_{it}:	7	4	4		7	8		8	1
Production day Y_{it}:	0	0	0		0	0		1	0
Amounts produced Z_1, Z_2:	0	8							

Child C after corrections

Trip list T with customers:	1	3	4	0	2	5	-1	2
Associated delivery day t:	1	1	1		1	1		2
Amounts supplied X_{it}:	7	4	7		4	9		8
Production day Y_{it}:	0	0	0		0	0		1
Amounts produced Z_1, Z_2:	0	8						

Fig. 2. Example of crossover.

$C \cdot T_R$ is corrected: customer 2 is not modified but customer 5 disappears because his demands satisfied in $B \cdot T$ are already satisfied in C. The production days coming from A (for $C \cdot T_M$) or B (for $C \cdot T_R$) require no modification because the production capacity and the storage capacity of the plant are both respected.

The complexity of one crossover is $O(n^2m^2)$. Indeed, in the worst case there are $O(nm)$ customer occurrences in $C \cdot T_M$ and all must moved to another day, with $O(nm)$ insertion positions.

3.5. Local search

Local search procedures for classical (single period) vehicle routing problems modify one or two trips at a time: for instance, one customer is removed from its current trip to be reinserted in another trip or in the same trip, but at a different position. The moves in the local search for the MA|PM are more complicated: they can change, for each customer in a day, the quantity delivered, the day of production, the day of delivery, the delivery trip and the position in this trip.

For instance, suppose that one delivery for a customer i is moved to another day t. As a customer cannot be visited more than once, the amount moved must be added to the one present in day t, if any. But this may be impossible if the receiving trip has not enough residual capacity. In such cases, the total amount for i may be moved to another trip, or a new trip may be created if some vehicles are not yet used. Of course, the production plan must be modified accordingly. This example illustrates the kinds of moves which must be enumerated, tested for feasibility and evaluated in terms of total cost variation.

The proposed method is a first-improvement local search applied to each day t of delivery and to each customer i visited that day. Two groups of moves are considered: in the same day t and from day t to another day u. In the sequel, $s(i)$ denotes the successor of client i in his trip.

In the first group, all pairs (i, j) of visited clients are considered, even if they belong to distinct trips, and the following moves are evaluated if vehicle capacity is respected:

- 2-OPT moves. If i and j are in the same trip, the subsequence $s(i) \rightarrow j$ is inverted. Otherwise, the shortest paths $(i, s(i))$ and $(j, s(j))$ are cut and replaced by either (i, j) and $(s(i), s(j))$ or by $(i, s(j))$ and $(j, s(i))$.
- Transfer i, i.e. remove i and reinsert it after j (j may be the depot-node for this move).
- Swap customers i and j.

Concerning the second group, the constraint of visiting each client at most once per day must be respected. We check first if the quantity delivered to client i on day t can be moved to another day u while respecting all constraints (storage capacity of customer and plant, transportation and production capacities, no shortages). It can be shown that possible destination days define two intervals of consecutive days before and after day t.

For each destination day u, there are two cases when moving to day u the amount delivered to customer i in day t: either i is already visited in day u, say by one trip R, or i is not visited in day u. In the first case, we first try to preserve the trip sequence, i.e. the amount X_{it} for day t is aggregated with the amount X_{iu} already delivered by R. If the aggregated amount violates the capacity of R, two kinds of moves are considered:

- Exchange i and his aggregated amount with another client j, in another trip of day u.
- Transfer the amount to another existing trip or to one new trip, if fleet size is respected.

There is one exception to the first-improvement rule here: the neighbourhood defined by the two last moves is explored completely, to achieve the best improvement.

In the second case (i not already visited in day u), we just try to insert i into an existing trip or into a new trip in day u, without exceeding the number of vehicles available.

To determine the complexity of one local search iteration (evaluation of moves to find the first improving one), note that each customer may appear in each day in the worst case. Hence, $O(mn)$ customers are inspected to be moved. For each of them, the moves in the same day can be evaluated in $O(n)$. If the customer is moved to another day u, the aggregation can be checked in $O(1)$, but if it fails, the best exchange or transfer for the aggregated amount costs $O(n)$. Since there are $O(m)$ possible days u, the complexity of one iteration is $O(m^2n^2)$. In practice, the local search is much faster because customers are seldom visited every day. Moreover, our implementation avoids testing moves uselessly. For instance, the possible insertions of a customer i in the trips of another day u are not evaluated if the residual fleet capacity in day u is not sufficient.

3.6. Distance measure

The distance measure $d(A, B)$ between two solutions A and B is the sum of two classical distances coming from error correcting codes: the Hamming distance d_H and the edit distance d_E, also called Levenshtein distance [41]. The Hamming distance is used to compare the vectors Z of the two solutions (amounts produced each day). It is usually defined on binary vectors but we extend it to count the number of days with a different status active/idle in A and B, i.e. $d_H = \sum_{t=1}^{m}((A \cdot Z_t = 0) \oplus (B \cdot Z_t = 0))$. The $\oplus$ operator denotes an exclusive *or* and it is assumed that the evaluation of the boolean expression returns 0 or 1.

The edit distance is very useful to evaluate the level of dissimilarity between two strings of symbols of different lengths, this is why we apply it to the trip lists $A \cdot T$ and $B \cdot T$ of A and B. It considers three elementary edit operations: replace one symbol in $A \cdot T$ by one from $B \cdot T$, delete one symbol in $A \cdot T$ and insert one symbol of $B \cdot T$ in $A \cdot T$. The distance d_E is nothing but the minimum number of operations to transform $A \cdot T$ into $B \cdot T$. It can be computed in $O((\mid A \cdot T \mid + \mid B \cdot T \mid)^2)$ by dynamic programming [41].

For instance, consider two strings $S = cdba$ and $S' = dbab$. S can be transformed into S' by deleting c and inserting b at the end. It is impossible to use less edit operations: the edit distance between S and S' is then $d_E(S, S') = 2$.

3.7. Algorithm overview

The memetic algorithm with population management can now be assembled using the components previously described. Its general structure is given by Algorithm 1. Six parameters explained in the sequel are used: *nc*, *maxcycles*, *maxcross*, π_{init}, π and *nkept*.

The method starts by generating an initial population P of *nc* random solutions (see 3.3). Each random solution S is improved by local search, with a given probability π_{init}. The distance measure is used to avoid duplicate solutions in P, i.e., S is added to P only if $d_P(S) \geqslant 1$. Indeed, both the Hamming and edit distances take integer values and $d_P(S) = 0$ if and only if S is already in P. The resulting population is sorted in ascending order of costs.

Algorithm 1 (*MA|PM overview*).

```
1: P ← ∅
2: for k =1 to nc
3:   repeat
4:     S ← random_solution()
5:     if random < π_init
6:       S ← local_search(S)
7:     end if
8:   until d_P(S) ⩾ 1
9:   P_k ← S
10: end for
```

11: sort P in ascending cost order
12: **for** $cycle = 1$ to $maxcycles$
13: initialize diversity threshold Δ
14: **for** $cross = 1$ to $maxcross$
15: select two parent-solutions A and B in P using binary tournament
16: $C \leftarrow crossover(A, B)$
17: **if** $random < \pi$
18: $C \leftarrow local\_search(C)$
19: **end if**
20: select R to be replaced in the worst half of P ($P(\lfloor nc/2 \rfloor)$ to $P(nc)$)
21: **if** $(D_{P\setminus\{R\}}(C) \geqslant \Delta)$ or $(cost(C) < cost(P_1))$
22: remove solution R: $P \leftarrow P \setminus \{R\}$
23: add solution C: $P \leftarrow P \cup \{C\}$
24: shift C to keep population P sorted
25: **end if**
26: update Δ
27: **end for**
28: $P \leftarrow partial\_renewal(P)$
29: **end for**
30: return the best solution (P_1)

The main loop performs a fixed number *maxcycles* of cycles and returns the best solution P_1 at the end. Each cycle initializes the diversity threshold Δ (see 3.1), executes a given number of basic iterations *maxcross* (innermost loop), modifies the population using a partial renewal procedure and adjusts Δ.

Each basic iteration chooses two parents A and B by binary tournament. The resulting child C undergoes the local search with a given probability π. The solution R to be replaced by C is pre-selected at random in the worst half of P (greatest costs). C is accepted if its distance to P minus R is at least Δ. Note that R is not included in the test because it will no longer be in P in case of replacement. To avoid missing a new best solution, C is also accepted if it improves the current best solution, i.e., if $cost(C) < cost(P_1)$. If C is accepted, it replaces R and a simple shift is sufficient to keep P sorted in increasing cost order. Otherwise C is simply discarded.

The rules to initialize Δ and to update it in each basic iteration depend on the control policies discussed in the sequel. Concerning the renewal procedure, it keeps the *nkept* best solutions and replaces the $nc - nkept$ others by "fresh" random solutions, improved by local search with probability π_{init}. Like in the initial population, all new solutions must be distinct.

3.8. Control policies for Δ

The simplest (static) policy is to use a constant diversity threshold Δ. For instance, $\Delta = 0$ corresponds to an ordinary MA in which clones (identical solutions) are allowed, while $\Delta = 1$ corresponds to an MA without clones like the one designed by Prins for the VRP [29]. Using larger values results in greater diversity but Δ must not be too large, otherwise most children are discarded and the MA|PM spends too much time in unproductive iterations.

Sörensen proposed in his Ph.D. thesis [33] what we call here the *up* and *down* policies, but without providing numerical results. These policies are based on a minimum value $\Delta_{\min}$, a maximum value $\Delta_{\max}$ and a step $\sigma = (\Delta_{\max} - \Delta_{\min})/maxcross$. The up-policy initializes Δ to $\Delta_{\min}$ and increments it by σ at each iteration, while the down-policy starts from $\Delta_{\max}$ and decrements by σ.

The last policy we have tried is a true dynamic one. It is similar to the up-policy, except that Δ is reset to $\Delta_{\min}$ each time the best solution P_1 is improved.

4. Computational evaluation

4.1. Implementation and instances

The methods designed in this paper were implemented in the Pascal-like programming language Delphi and tested on a 2.8 GHz PC under Windows XP.

To the best of our knowledge, no instances are publicly available for our problem. This is why we reuse here the randomly generated instances used to evaluate a GRASP in a previous paper [3]. There are three sets of 30 instances with 50, 100 and 200 customers, all with a planning horizon of $m = 20$ periods. It is important to note that a VRP (single-period problem) with 200 customers is already considered as a large-scale problem which cannot be solved to optimality by existing exact algorithms. Adding a 20-period horizon makes the problem even more combinatorial. In any case, our instances are much bigger than the ones used by Chandra and Fisher [8].

The random generation of some data in the instances depends on the number of customers: production capacity, customers and plant storage capacities, number of vehicles and setup cost. On average, the production capacity PPC represents 3–4 days of customer consumption and the fleet capacity fW two days of consumption for the whole set of customers. The production setup cost s is proportional to production capacity. The storage capacity PSC at the plant varies between one and a half and twice the production capacity. The holding cost h at the plant is the same for all instances.

4.2. Algorithms compared and parameters used

The design of tight lower bounds for the IPDP seems a very hard task: the existing bounds for the VRP are already extremely sophisticated and based on complex branch-and-cut algorithms and, to the best of our knowledge, no lower bound has ever been published for the Periodic VRP. This is why the MA|PM was compared with two algorithms briefly summarized in Section 2: the two-phase heuristic H1 of [2], which handles production and distribution separately, and the GRASP metaheuristic of [3], which tries to coordinate production and distribution decisions.

Table 1
Impact of some parameters

Version	π_{init}	π	Policy	*maxcycles*	*maxcross*	Gain on H1 (%)	Time (seconds)
1	0	0.2	Up	4	2500	23.03	172.7
2	0.5	0.5	Up	4	2500	22.21	200.7
3	0	0.2	Up	5	2000	21.72	195.6
4	1	0.2	Up	4	2500	22.55	182.2
5	0.5	0.2	Down	4	2500	22.99	269.4
6	0	0.2	Down	4	2500	22.75	255.3
7	1	0.2	Down	4	2500	22.90	284.4
8	0	0	Static $\Delta_{\min}$	4	2500	15.95	197.18
9	0	0.2	Static $\Delta_{\min}$	4	2500	22.48	199.08
10	0	0	Static $\Delta_{\max}$	4	2500	17.85	192.02
11	0	0.2	Static $\Delta_{\max}$	4	2500	21.91	201.14

Table 2
Results for instances with 50 customers

Instance	Heuristic H1		GRASP			MA|PM		
	Cost	Time (seconds)	Cost	Time (seconds)	Saving	Cost	Time (seconds)	Saving
1	487,835	0.16	440,630	51.16	09.68	378,378	170.35	22.44
2	517,636	0.14	465,005	88.42	10.17	403,913	149.02	21.97
3	510,862	0.09	455,618	98.07	10.81	409,573	135.72	19.83
4	519,366	0.09	456,518	56.13	12.10	399,220	159.66	23.13
5	516,549	0.11	434,686	65.65	15.85	422,279	193.34	18.25
6	554,784	0.10	452,639	90.73	18.41	407,122	174.74	26.62
7	498,413	0.10	436,677	110.62	12.39	414,977	173.98	16.74
8	506,887	0.09	422,583	77.34	16.63	379,744	170.83	25.08
9	497,002	0.10	434,849	133.81	12.51	407,935	158.09	17.92
10	534,695	0.10	454,163	121.12	15.06	396,258	179.44	25.89
11	510,978	0.10	434,055	72.79	15.05	402,475	178.25	21.23
12	539,599	0.11	453,140	74.49	16.02	358,702	151.02	33.52
13	495,836	0.09	440,891	97.08	11.08	371,030	193.33	25.17
14	494,893	0.08	433,018	79.90	12.50	406,114	160.81	17.94
15	490,994	0.09	454,100	90.08	07.51	373,076	173.41	24.02
16	496,658	0.10	457,911	90.51	07.80	379,404	186.22	23.61
17	531,234	0.10	455,878	91.08	14.19	406,353	177.52	23.51
18	572,669	0.12	442,015	88.53	22.81	401,179	163.54	29.95
19	502,135	0.09	461,571	99.39	08.08	406,893	186.93	18.97
20	511,517	0.10	463,798	91.25	09.33	398,508	182.72	22.09
21	487,970	0.10	410,112	66.23	15.96	397,112	188.36	18.62
22	475,653	0.10	429,296	83.17	09.75	358,749	146.07	24.58
23	524,725	0.11	455,882	95.90	13.12	407,369	188.49	22.37
24	503,746	0.09	430,242	101.40	14.59	369,784	180.11	26.59
25	531,122	0.09	462,605	76.65	12.90	411,556	192.38	22.51
26	500,779	0.09	444,321	75.03	11.27	408,704	159.93	18.39
27	489,330	0.09	444,984	82.28	09.06	366,197	173.13	25.16
28	488,786	0.09	450,254	104.55	07.88	401,032	171.03	17.95
29	532,026	0.10	464,111	67.52	12.77	384,282	198.35	27.77
30	522,688	0.09	433,901	88.39	16.99	369,959	163.48	29.22
Average	511,578.9	0.10	445,848.4	86.97	12.74%	393,262.5	172.68	23.03%

Some preliminary testing was required to tune the parameters, using the instances with 50 customers. Three parameters were quickly fixed: the population size *nc*, the number of solutions preserved in the partial renewal procedure *nkept* and the mininimum diversity threshold $\Delta_{\min}$. As underlined by Sörensen [33], MA|PM requires a small population to avoid excessive rejection rates. We took $nc = 20$ but the results are stable between 15 and 30. Varying *nkept*, the best results are always obtained if all solutions except the best one are replaced in the partial renewal procedure, i.e. with $nkept = 1$. $\Delta_{\min}$ was set to 1 to forbid clones in all cases.

Several values were tested for each other parameter. The local search rate π_{init} on the initial population was set to 0%, 50% or 100%. As observed in [30,31], the local search rate π after each crossover must be smaller, to avoid too many failures in the distance test. Two values were evaluated: 20% and 50%. Concerning the MA duration, better results are obtained if the total number of crossovers is

partitioned into several cycles, but cycles must be long enough otherwise the algorithm has no time to converge in each cycle. We took finally $maxcross = 10,000$ crossovers in total and $maxcycles \in \{4, 5\}$, i.e. four cycles of 2500 crossovers or five cycles of 2000.

The static, up, down and dynamic policies were compared, using several constant values for Δ in the static policy. The other policies were tested with $\Delta_{\min} = 1$ and $\Delta_{\max} = \overline{\Delta}/2$, where $\overline{\Delta}$ is the average inter-solution distance in the initial population *not improved* by local search ($\pi_{init} = 0$). This average distance is 5 times smaller in an initial population improved by local search ($\pi_{init} = 1$). A larger $\Delta_{\max}$ does not yield better results but increases running times, due to rejection rates exceeding 40%. With smaller values, the diversity criterion defined by the distance function has little impact and the algorithm is not really better than a classical MA without clones.

Only a few efficient settings are reported in Table 1, to avoid listing all combinations. The performance criteria are the gain in % compared to the two-phase heuristic H1 and the running time in seconds. The best gain and running time are obtained by version 1, with an initial population not improved by local search, a moderate local search rate during the iterations ($\pi = 0.2$), the up-policy and four cycles of 2500 crossovers each. The other lines are given to show the impact of each parameter. Compared to version 1, the other versions based on the up-policy (2–4) show degraded results with a local search on the initial population, a higher local search rate during the algorithm, or a partition of the crossover into smaller cycles.

The down-policy is used in versions 5–7, which are slower than the algorithms with the up-policy. The reason is an important percentage of discarded children at the beginning of the MA, when the population is not yet well exploited. In spite of this drawback, version 5 is almost as good as version 1 and versions 6 and 7 are only a bit inferior. Moreover, the variants with the down-policy are little affected by the local search rate applied to initial solutions.

Concerning the static policy, versions 8–9 use a primitive population management: clones are forbidden. Version 8 without local search corresponds to a basic GA and version 9 to an ordinary MA. Versions 10–11 apply a constant but high diversity thereshold. All these variants are dominated by version 1.

Table 3
Results for instances with 100 customers

Instance	Heuristic H1		GRASP			MA|PM		
	Cost	Time (seconds)	Cost	Time (seconds)	Saving	Cost	Time (seconds)	Saving
1	954,888	0.47	796,183	375.09	16.62	714,401	1147.4	25.18
2	970,819	0.52	785,228	542.21	19.12	722,047	1192.8	25.62
3	956,994	0.55	790,768	395.20	17.37	677,598	925.9	29.20
4	954,752	0.51	782,130	524.45	18.08	710,552	1097.3	25.58
5	969,945	0.55	799,622	439.06	17.56	733,040	1125.0	24.42
6	947,350	0.51	794,960	429.60	16.09	696,146	1107.0	26.52
7	941,940	0.53	781,909	427.33	16.99	705,322	1081.8	25.12
8	950,303	0.53	790,804	487.45	16.78	679,210	1043.1	28.53
9	960,867	0.55	763,180	429.37	20.57	699,518	1136.1	27.20
10	963,192	0.53	792,321	366.27	17.74	705,778	976.9	26.73
11	960,584	0.53	790,507	400.86	17.71	709,122	1154.7	26.18
12	967,656	0.55	798,520	316.64	17.48	755,726	1163.3	21.90
13	961,435	0.52	778,341	513.70	19.04	695,466	1036.6	27.66
14	957,707	0.53	792,644	467.77	17.24	718,260	1153.7	25.00
15	958,347	0.55	795,775	454.45	16.96	736,041	1252.7	23.20
16	961,510	0.53	799,755	393.74	16.82	715,209	1210.9	25.62
17	979,473	0.53	800,484	510.40	18.27	737,832	964.5	24.67
18	968,923	0.53	788,922	381.72	18.58	723,413	1021.0	25.34
19	1,006,653	0.61	80,7291	298.10	19.80	720,218	1164.4	28.45
20	974,115	0.53	811,026	347.59	16.74	724,727	1167.4	25.60
21	959,144	0.53	794,644	359.96	17.15	724,328	1173.5	24.48
22	980,325	0.52	806,817	390.68	17.70	701,506	1179.8	28.44
23	946,747	0.62	784,617	308.83	17.12	710,033	1196.9	25.00
24	974,608	0.61	798,762	431.23	18.04	734,327	1250.7	24.65
25	958,907	0.53	796,871	364.84	16.90	725,446	848.4	24.35
26	966,225	0.55	796,232	384.81	17.59	718,939	1094.4	25.59
27	959,204	0.53	784,542	464.78	18.21	715,068	1033.7	25.45
28	967,618	0.53	782,572	416.89	19.12	685,117	1092.3	29.20
29	970,083	0.55	804,090	457.51	17.11	722,571	1069.8	25.51
30	959,149	0.55	785,952	396.56	18.06	721,850	1180.3	24.74
Average	963,648.8	0.54	792,515.6	415.90	17.75%	714,627	1108.1	25.84%

4.3. Results

The other tables provide the results obtained by the best version of the MA|PM (version 1 of Table 1) on the three sets of instances and compare them with the two-phase method H1 and the GRASP. They indicate solution values and running times for each algorithm and, for the GRASP and MA|PM, the saving obtained from H1.

Table 2 concerns the benchmark problems with 50 customers. Recall that H1 represents the two-phase approach used by most companies: vehicle trips are determined daywise after the production planning phase. The GRASP and the MA|PM are expected to give better results because they handle production and distribution decisions simultaneously, for instance their local searches can change both the delivery day and the production day of each demand.

This is confirmed by the average results in the table. The only interest of H1 is its speed (0.1 seconds on average): the saving and the running time are 12.74% and 86.97 seconds for the GRASP, and 23.03% and 172.68 seconds (almost 3 minutes) for the MA. The gain obtained with the GRASP is already important and confirms the interest of a coordinated approach. However, the weak point of this metaheuristic is the independency of its iterations, which constitute a kind of random sampling of solution space. The MA is more effective because it works in parallel on a population of high-quality solutions. Even instance per instance, it always outperforms the GRASP.

The trends observed on Table 2 are confirmed in Table 3 for instances with 100 customers. On average, the GRASP approximately saves 17.75% in 415.90 seconds of CPU time, versus 25.84% in 1108.1 seconds (almost 19 minutes) for the memetic algorithm. Here again, the MA|PM is always better than the GRASP: its minimum saving is 21.90% (instance 12), while the *maximum* saving achieved by the GRASP is 20.57% on instance 9. However, both metaheuristics realize better savings compared to the instances with 50 customers: on the first set of instances, the local search finds less feasible and improving moves.

Results for 200 customers are listed in Table 4. The savings of GRASP and MA are here a bit smaller, 17.29% and 23.64%. This is also the only group of instances for which the GRASP does better than the MA, but only for instance 10: 15.51 versus 14.98%.

The running times become important: 1801.80 seconds for the GRASP (approximately 30 minutes) and 4098.5

Table 4
Results for instances with 200 customers

Instance	Heuristic H1		GRASP			MA|PM		
	Cost	Time (seconds)	Cost	Time (seconds)	Saving	Cost	Time (seconds)	Saving
1	1,321,885	2.03	1,090,955	1802.70	17.47	996,151	3633.78	24.64
2	1,309,448	2.06	1,085,245	2002.73	17.12	978,373	3755.53	25.28
3	1,311,514	2.06	1,095,374	1419.91	16.48	986,147	3629.63	24.81
4	1,285,302	2.00	1,083,718	1908.09	15.68	962,937	3851.76	25.08
5	1,280,850	2.03	1,080,953	1326.52	15.61	970,638	4185.51	24.22
6	1,302,811	2.16	1,077,213	2101.69	17.32	965,646	3794.83	25.88
7	1,313,975	2.00	1,107,547	1631.94	15.71	980,562	4732.91	25.37
8	1,371,367	2.36	1,064,031	2237.27	22.41	1,014,809	3929.44	26.00
9	1,286,548	1.99	1,079,408	1549.83	16.10	967,738	3424.77	24.78
10	1,285,798	2.03	1,086,390	1804.78	15.51	1,093,230	4600.79	14.98
11	1,310,551	2.14	1,092,441	1850.03	16.64	1,008,080	5440.02	23.08
12	1,319,198	2.23	1,076,313	1857.58	18.41	998,951	3933.37	24.28
13	1,391,792	2.26	1,090,483	2087.02	21.65	984,918	4662.04	29.23
14	1,390,506	2.33	1,090,927	2103.95	21.54	964,301	3596.09	30.65
15	1,304,868	2.08	1,080,625	1751.50	17.19	981,167	4023.73	24.81
16	1,285,485	2.06	1,084,519	1609.41	15.63	1,017,777	4245.70	20.83
17	1,302,606	2.00	1,084,039	1746.55	16.78	1,073,640	4355.35	17.58
18	1,387,248	2.24	1,109,761	1684.84	20.00	1,003,670	3875.22	27.65
19	1,308,273	2.09	1,078,144	2143.42	17.59	997,348	4157.69	23.77
20	1,284,225	2.03	1,074,537	1764.66	16.33	981,788	4048.08	23.55
21	1,308,456	2.12	1,079,881	2422.88	17.47	974,384	4205.02	25.53
22	1,284,313	2.05	1,083,007	1681.44	15.67	1,065,780	4465.87	17.02
23	1,321,233	2.05	1,083,438	1761.78	18.00	1,070,520	3147.22	18.98
24	1,301,522	2.19	1,081,914	1592.80	16.87	978,491	3308.83	24.82
25	1,289,588	2.02	1,090,172	1739.25	15.46	1,029,327	4349.00	20.18
26	1,307,163	2.02	1,066,180	1685.31	18.44	961,728	3785.77	26.43
27	1,318,022	2.11	1,111,924	1517.36	15.64	1,028,006	4086.87	22.00
28	1,294,872	2.13	1,059,007	1508.17	18.22	1,011,689	4456.09	21.87
29	1,285,875	2.17	1,103,486	1522.44	14.18	1,015,741	5168.01	21.01
30	1,313,075	2.13	1,080,461	2238.44	17.72	985,496	4105.80	24.95
Average	1,312,612.3	2.10	1,085,069.7	1801.80	17.29%	1,001,634.4	4098.5	23.64%

Table 5
Gain from H1 in % and time in seconds for GRASP versions

n	GRASP		RGRASP		GRASP-PR1		GRASP-PR2		MA|PM	
	Gain	Time	Gain	Time	Gain	Time	Gain	Time	Gain	Time
50	12.74	86.97	13.25	93.45	13.41	87.90	13.49	131.49	23.03	172.7
100	17.75	415.90	17.83	415.60	17.99	436.42	18.02	466.22	25.84	1108.1
200	17.29	1801.80	18.45	1893.75	18.40	1846.69	18.29	2116.86	23.64	4098.5

seconds for MA|PM (68 minutes). However, it is important to note that the IPDP concerns the tactical decision level of companies: such running times are reasonable because the algorithms do not need to be executed every day. Moreover, the IPDP must be compared with the VRP and the PVRP for a correct appraisal of running times. For instance, Cordeau et al. have recently published a survey of best metaheuristics for the single-period VRP [13] and report typical running times of 15 minutes for 200–250 customers on a 2 GHz PC. Since the IPDP distribution subproblem is a much more combinatorial PVRP, our computational times of 68 minutes for 200 customers but 20 periods are not excessive.

A natural question is to wonder whether the solution gap between the GRASP and the MA could be reduced by increasing the number of GRASP iterations (500). We tried to increase this number of iterations for the three sets of instances, to have the same running time for the two algorithms, but no significant improvement was obtained. We tested also the configuration giving to MA|PM approximatively the same running time as GRASP. The results obtained show that the MA|PM is still better. For example, the average saving obtained for the set of instances with 50 customers is 20.02% and this for an average running time of 100.24 seconds versus 12.74% and 86.97 seconds for the GRASP. To reduce the MA|PM running time, two cycles of 2500 crossovers were executed instead of four.

Due to lack of space, the results of the other GRASPs described in [3] cannot be given instance per instance but Table 5 gives their average saving from H1 in % and their average running time in seconds, for each instance set. RGRASP is a reactive version, in which the length of the restricted candidate list is dynamically adjusted during the search. The two GRASP + PR are reinforced by a path relinking (PR) procedure. Path relinking consists of exploring trajectories that link a few elite solutions in solution space. The PR1 version uses the PR as a post-optimization technique, after the GRASP. In the PR2 variant, the PR is applied in each GRASP iteration, to explore the path between the local optimum found and one elite solution.

Table 5 shows that the reactive GRASP and the two versions with path relinking outperform the basic GRASP. However, the maximum improvement, brought by the reactive GRASP on the 200-customer problems, is limited to 1.16% and the price to pay is an increased running time. In all cases, the MA|PM is much better: compared to the best GRASP version, it obtains an additional saving between 5.19% (200 customers) and 9.54% (50 customers).

The three-phase heuristic H2 mentioned in Section 2, which implements a weaker integration between production and distribution, takes place between H1 and the basic GRASP. Its saving to H1 range from 9% for 50 customers to 13% for 100 and 200 customers.

5. Conclusion and future directions

In this paper, the very recent MA|PM framework is applied to a combined production–distribution problem. This requires the definition of non-trivial components: an ad-hoc encoding, a crossover operator, an effective local search able to change both the production plan and the distribution plan, and a distance measure in solution space.

The tests show that the resulting algorithm can tackle large instances (200 customers and 20 periods) in reasonable amounts of time. The savings (23% or more) are important compared to a classical approach in which the production plan is determined before the vehicle routing phase. The MA|PM is even better than a sophisticated GRASP in which production and distribution decisions are also considered globally. Our metaheuristic achieves these results with a relatively simple policy for the diversity threshold, the up-policy. Dynamic policies are more complicated but do not yield better results. The versions with $\Delta = 0$ or 1, which correspond to classical memetic algorithms without population management, are inferior.

The main interest of our model and algorithms for company managers is a better coordination between production and distribution, leading to a better resource utilization (production lines, storage areas, vehicles) and a significant reduction of the overall system cost. Moreover, two software applications (one for production planning and one for vehicle routing) could be replaced by a single one. Of course, further research is still necessary to integrate complications met in practice, like several products, stochastic demands or time windows to visit customers.

Among other possible future directions, we would like to confirm the efficiency of the MA|PM, compared to other metaheuristics like Scatter Search or Tabu Search. To avoid the design of experiments used to tune the parameters, self-adaptive memetic algorithms [35] could be developed.

Our research on the multi-product case is in progress, with some preliminary results using a cooperative method [1]. Each iteration of this method performs two phases. Phase 1 solves the production planning problem, modelled as an integer linear program. In phase 2, vehicle trips are computed in each period. The results of phase 2 are used

to modify the production plan in the first phase of the next iteration, and so on. The production plan and the distribution plan become quickly stable, and a perturbation procedure is required in practice to avoid a premature convergence.

References

[1] M. Boudia, S. Dauzère-Pérès, M.A.O Louly, C. Prins. Integrated optimization of production and distribution, in: ICSSSM'06 IEEE Int. Conf. on Service Systems and Service Management, IEEE, 2006, pp. 272–276.

[2] M. Boudia, M.A.O. Louly, C. Prins. Combined optimization of production and distribution. In: International Conference on Industrial Engineering and Systems Management (IESM′05, Marrakech, Morocco), 10 pages on CD. I4E2, Mons, Belgium, 2005. Extended version to appear in Production Planning and Control.

[3] M. Boudia, M.A.O. Louly, C. Prins, A reactive GRASP and path relinking for a combined production-distribution problem, Computers and Operations Research 34 (2006) 3402–3419.

[4] J. Bramel, S. Goyal, P. Zipkin, Coordination of production/distribution networks with unbalanced leadtimes, Operations Research 48 (2000) 570–577.

[5] A.M. Brewer, K.J. Button, D.A. Hensher, Handbook of Logistics and Supply Chain Management, Pergamon, Oxford, 2001.

[6] E.S. Buffa, R.K. Sarin, Modern Production/Operations Management, Wiley, New York, 1987.

[7] P. Chandra, A dynamic distribution model with warehouse and customer replenishment requirements, Journal of the Operational Research Society 44 (1993) 681–692.

[8] P. Chandra, M.L. Fisher, Coordination of production and distribution planning, European Journal of Operational Research 72 (1994) 503–517.

[9] K.J. Chen, P. Ji, A mixed integer programming model for advanced planning and scheduling (APS), European Journal of Operational Research 181 (1) (2007) 515–522.

[10] M. Christiansen, B. Nygreen, A method for solving ship routing problems with inventory constraints, Annals of Operations Research 81 (1998) 357–378.

[11] N. Christofides, J.E. Beasley, The period routing problem, Networks 14 (1984) 237–256.

[12] G. Clarke, J.W. Wright, Scheduling of vehicles from a central depot to a number of delivery points, Operations Research 12 (1964) 568–581.

[13] J.F. Cordeau, M. Gendreau, A. Hertz, G. Laporte, J.S. Sormany, New heuristics for the vehicle routing problem, in: A. Langevin, D. Riopel (Eds.), Logistic Systems: Design and Optimization, Wiley, 2005, pp. 279–298.

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

[15] J.A. Ferland, S. Ichoua, A. Lavoie, E. Gagné, Scheduling using tabu search methods with intensification and diversification, Computers and Operations Research 28 (2001) 1075–1092.

[16] R. Fukasawa, J. Lysgaard, M. Poggi de Aragão, M. Reis, E. Uchoa, R.F. Werneck, Robust branch-and-cut-and-price for the capacitated vehicle routing problem, Mathematical Programming 106 (2006) 491–511.

[17] F. Fumero, C. Vercellis, Development of production, inventory, and distribution schedules, Transportation Science 33 (1999) 330–340.

[18] M. Gendreau, G. Laporte, R. Séguin, Stochastic vehicle routing, European Journal of Operational Research 88 (1996) 3–12.

[19] B.R. Golden, E.A. Wasil, Computerized vehicle routing in the soft drink industry, Operations Research 35 (1987) 6–17.

[20] S.C. Graves, Manufacturing planning and control, in: P. Pardalos, M. Resende (Eds.), Handbook of Applied Optimization, Oxford University Press, New York, 2002, pp. 728–746.

[21] S.C. Graves, A.H.G. Rinnooy Kan, P.H. Zipkin, Handbooks in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory, Elsevier, North-Holland, 1993.

[22] A. Hertz, M. Widmer, Guidelines for the use of meta-heuristics in combinatorial optimization, European Journal of Operational Research 151 (2) (2003) 247–252.

[23] M. Lacomme, C. Prins, W. Ramdane-Chérif, Competitive memetic algorithms for arc routing problems, Annals of Operations Research 131 (2004) 159–185.

[24] M. Laguna, R. Martí, V. Campos, Intensification and diversification with elite tabu search solutions for the linear ordering problem, Computers and Operations Research 26 (1999) 1217–1230.

[25] R.D. Metters, Interdependent transportation and production activity at the United States postal service, Journal of the Operational Research Society 47 (1996) 27–37.

[26] P. Moscato, Memetic algorithms: a short introduction, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, 1999, pp. 219–234.

[27] S. Nahmias, Productions and Operations Analysis, Irwin/McGraw-Hill, USA, 1997.

[28] C. Prins, Efficient heuristics for the heterogeneous fleet multitrip vehicle routing problem, Journal of Mathematical Modelling and Algorithms 1 (2) (2002) 135–150.

[29] C. Prins, A simple and effective evolutionary algorithm for the vehicle routing problem, Computers and Operations Research 31 (2004) 1985–2002.

[30] C. Prins, C. Prodhon, R. Wolfler Calvo, A memetic algorithm with population management (MA|PM) for the capacitated location-routing problem, in: J. Gottlieb, G.R. Raidl (Eds.), Evolutionary computation in combinatorial optimization, volume 3906 of Lecture Notes in Computer Science, Springer, 2006, pp. 183–194.

[31] C. Prins, M. Sevaux, K. Sörensen. A genetic algorithm with population management (GA|PM) for the CARP, in: 5th Triennal Symposium on Transportation Analysis (Tristan V), Le Gosier, Guadeloupe, 2004.

[32] A.M. Sarmiento, R. Nagi, A review of integrated analysis of production distribution systems, IIE Transactions 31 (1999) 1061–1074.

[33] K. Sörensen, A Framework for Robust and Flexible Optimisation Using Metaheuristics with Applications in Supply Chain Design. Ph.D. thesis, University of Antwerp, Belgium, 2003.

[34] K. Sörensen, M. Sevaux, MA|PM: Memetic algorithms with population management, Computers and Operations Research 33 (2006) 1214–1225.

[35] J. Tang, M.H. Lim, Y.S. Ong, Adaptation for parallel memetic algorithms based on population entropy, in: M. Cattolico (Ed.), GECCO 2006, ACM, 2006, pp. 575–582.

[36] J. Tang, M.H. Lim, Y.S. Ong, Diversity-adaptive parallel memetic algorithm for solving large scale combinatorial optimization problems, Soft Computing Journal 11 (9) (2007) 873–888.

[37] J. Tang, M.H. Lim, Y.S. Ong, M.J. Er, Parallel memetic algorithm with selective local search for large scale quadratic assignment problems, International Journal of Innovative Computing, Information and Control 2 (6) (2006) 1399–1416.

[38] P. Toth, D. Vigo, The Vehicle Routing Problem, SIAM, Philadelphia, 2002.

[39] A. Turano. Joint Production, Distribution ans Inventory Planning: Models ans Solution Approaches for Supply Chain Coordination. PhD thesis, Universita Degli Studi di Brescia, Brescia, Italy, 2005.

[40] S. Voss, D.L. Woodruff, Introduction to Combinatorial Optimization Models for Production Planning in a Supply Chain, Springer, Berlin, 2003.

[41] R.A. Wagner, M.J. Fischer, The string-to-string correction problem, Journal of the Association for Computing Machinery 21 (1974) 168–173.