

Evolutionary Algorithms for Multiperiod Arc Routing Problems

Philippe Lacomme
LIMOS
Université Blaise Pascal
Campus des Cézeaux
63177 Aubière Cedex, France
lacomme@sp.isima.fr

Christian Prins
LOSI
Univ. de Technologie de Troyes
12 Rue Marie Curie, BP 2060
10010 Troyes Cedex, France
prins@utt.fr

Wahiba Ramdane-Chérif
LOSI
Univ. de Technologie de Troyes
12 Rue Marie Curie, BP 2060
10010 Troyes Cedex, France
ramdane@utt.fr

Abstract

The Capacitated Arc Routing Problem (CARP) is useful for modeling urban waste collection or winter gritting, at the operational decision level. Many applications require a tactical version, the periodic CARP (PCARP), in which the trips are planned over a multiperiod horizon, subject to frequency constraints. The paper presents this new and hard theoretical problem, its variants, and a powerful genetic algorithm (GA) tackling the tactical and operational levels in parallel. Preliminary tests with the GA show important savings compared to one constructive heuristic and another method that assigns tasks to periods before solving the CARP in each period.

Keywords: vehicle routing, CARP, multiperiod planning problem, genetic algorithm.

1 Introduction

Contrary to the well-known *Vehicle Routing Problem* (VRP), in which client nodes in a network must be visited for pickup or delivery, the *Capacitated Arc Routing Problem* (CARP) consists of processing a subset of arcs. CARP applications include for instance urban waste collection, winter gritting or inspection of power lines. From now on, without loss of generality, we take our examples in municipal refuse collection to make the paper more explanatory.

The basic CARP of the literature is a *single-period* and *undirected* model. Each edge in the network may be processed in any direction and corresponds to a 2-way street whose both sides can be collected during one traversal (zigzag collection), a frequent situation in low-traffic suburban areas. A fleet of identical vehicles of

limited capacity is based at a depot node. Each edge can be traversed any number of times. It has a traversal cost and a demand. A subset of edges require service by a vehicle. The CARP consists of determining a set of vehicle trips of minimum total cost, such that each trip starts and ends at the depot, each required edge is serviced by one single trip, and the total demand for each trip does not exceed vehicle capacity.

The CARP is NP-hard. Two typical heuristics to get good solutions are Path-Scanning [3] and Augment-Merge [4], both proposed by Golden and Wong. Excellent solutions can be obtained by recent metaheuristics like the tabu search method from Hertz *et al.* [5] and the very efficient genetic algorithm (GA) proposed by Lacomme, Prins and Ramdane-Chérif [6].

The basic CARP is not realistic enough for practical applications like waste collection. First, real street networks are *mixed*: they comprise edges, but also independent arcs representing street lanes whose direction must be respected. Second, trips need to be planned over a *multiperiod time horizon*.

Section 2 shows how to handle mixed graphs and prohibited turns for the single-period case. It also recalls the principles of the GA published in [6] for this augmented CARP. Section 3 adds the multiperiod horizon, giving a new theoretical problem called PCARP (Periodic CARP). It also explains how to deal with PCARP complications like varying demands and spacing constraints. Section 4 presents a non-trivial extension of the single-period GA to the PCARP. Section 5 is devoted to a computational evaluation that compares two versions of the new GA with two other heuristics.

2 Single-period arc routing

2.1 An extended CARP for mixed networks

We describe below notations and data structures for a more realistic CARP based on a mixed network with prohibited turns. The network must be encoded as an *entirely directed graph* $G = (N, A)$, with a set N of nn nodes (crossroads) and a set A of na arcs (lanes with traffic directions). N includes a depot node with nva vehicles of capacity Q (nva is either fixed or left as a decision variable). The classical notation of arcs as pairs of nodes becomes ambiguous in case of multiple lanes and prohibited turns, e.g., the shortest path from a node i to a node j depends on the arcs used to reach i and leave j . We then prefer to use arc indexes from 1 to na .

Each arc u begins at a node $b(u)$, ends at a node $e(u)$ and can be traversed any number of times at cost $c(u)$ (e.g., a duration or distance). A subset R of nra arcs require service by a vehicle. Each required arc u has a demand $r(u)$ and a service cost $w(u)$. All costs and demands are non-negative integers. The arcs in R represent nt tasks for the vehicles: *net* edge-tasks (pairs of opposite arcs of R , serviced together, in any direction) and *nat* arc-tasks (independent arcs of R). Hence, $nra = nat + 2 \cdot net$. Two arcs u, v for the same edge are linked with a pointer inv such that $inv(u) = v$ and $inv(v) = u$. Moreover, $r(u) = r(v)$ is the total demand on the edge and $w(u) = w(v)$.

By convention, we set $r(u)$ and $c(u)$ to 0 if u is not required. This allows for instance to get the total demand by a simple sum over A . We also set $inv(u)$ to 0 if u does not belong to an edge, whether u is required or not.

The graph is defined by giving for each arc u a list $succ(u)$ of allowed successor-arcs: $v \in succ(u)$ if arcs u and v are consecutive ($e(u) = b(v)$) and if it is permitted to turn from u to v . The shortest paths are given by a matrix D , $nra \times nra$. For any two arcs u and v , $d(u, v)$ is the traversal cost of a shortest path from u to v (*not included*, to ease operations on trips like arc insertions), taking prohibited turns into account. Paths from or to the depot are handled by including a fictitious loop in A for the depot. D can be computed by adapting Dijkstra's shortest path algorithm [2]. A trip can be stored as a list of required arcs, assuming shortest paths between successive tasks and between a task and the depot loop. Prohibited turns become transparent.

2.2 A GA for the extended CARP

The GA of [6] is briefly recalled here, because its chromosomes and evaluation procedure are partly reused in section 4 for the PCARP. Consider one solution to the extended CARP. Its chromosome is built by concatenating the list of tasks of each trip. This gives a sequence of nt required arcs in which each task occurs once (an edge can occur as one of its two opposite arcs). It can be interpreted as a long tour performed by one single vehicle of infinite capacity, servicing all tasks. This coding is appealing because *there always exists one optimal sequence*.

Chromosomes have no trip delimiters: the trips and their total cost are computed by an *optimal method SPLIT* (Figure 1). The upper part shows a long tour of 4 (thick) edge-tasks a, b, c, d in an undirected context, with demands in brackets. Traversal and service costs are equal. Thin lines are possible shortest paths linking successive tasks or a task and the depot. In the middle, an auxiliary acyclic graph H is built with one arc per possible trip, assuming a vehicle capacity $Q=9$: e.g., arc ab stands for a trip servicing a, b and costing 51. A shortest path in H (thick line) can be computed using Bellman's algorithm [2]. It gives the optimal splitting for the chromosome (here two trips, for a total cost 115).

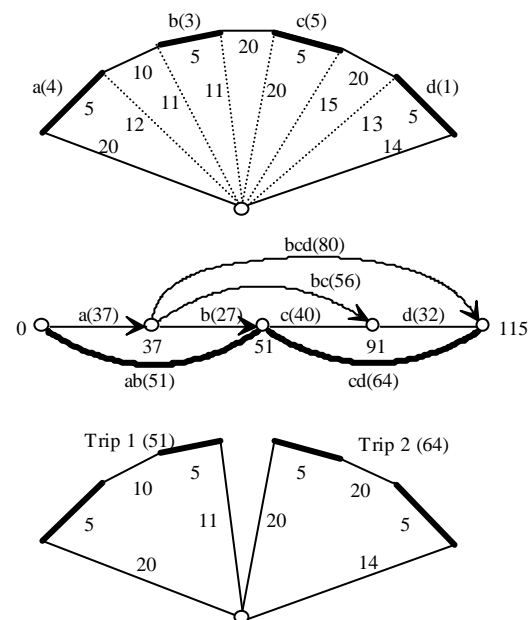


Figure 1: Evaluation procedure SPLIT

Thanks to the chromosomes without trip limits, classical crossovers like OX and LOX can be applied to make children that can be evaluated optimally by the splitting procedure. Searching

the best sequence is left to the intrinsic parallelism of the GA. The reader will find a description of the other GA components in [6].

This GA is currently the most efficient solution method for the CARP. Solution quality can be evaluated with a tight lower bound proposed by Belenguer and Benavent [1]. For instance, on a set of 34 basic CARP benchmarks generated by Belenguer, 22 are solved to optimality (the bound is reached) and the average distance to the bound is only 0.69%. In comparison, the tabu search from Hertz (the best method before our GA) finds the optimum for 15 instances, with an average deviation of 1.9%. The excellent behavior of the GA was the main reason to envisage genetic algorithms for solving multiperiod arc routing problems.

3 Planning problems in arc routing

3.1 The basic periodic CARP

Consider a time horizon H of np periods or "days", in which each task u has a *frequency* $f(u)$, with $f(u) = f(inv(u))$ for an edge-task. This means that task u must be serviced $f(u)$ times, $1 \leq f(u) \leq np$, but at most once per day: for instance, the waste collecting trips are defined in France for one or two weeks. The total number of services ns to be performed over H is then:

$$(1) \quad ns = \sum_{u \in T} f(u)$$

The *arc routing planning problem* or *periodic CARP (PCARP)* consists of assigning each task u to $f(u)$ distinct days to minimize the total cost of the trips over H . The PCARP is obviously NP-hard because it includes the basic (single-period) CARP as particular case.

Compared to production management, the assignment phase corresponds to a production planning problem (*tactical* decision level), while the detailed solution of the resulting CARP for one day corresponds to a scheduling problem (*operational* decision level). In production planning, schedules are often computed for the first period of H only, because other periods are still too uncertain. Then H is shift by one period, and the new first period can be detailed.

In most arc routing problems like waste collection, this *rolling horizon* technique is not valid because service days for each street must be identical for all weeks: people would forget to get their bins out if service days change too

often. However, it becomes now possible to *solve simultaneously the planning and scheduling levels*, since all scheduling problem (daily CARPs) are defined if service days are known. The GA of section 4 exploits this possibility.

Unfortunately, in some applications like waste collection, the basic PCARP is complicated by demands that fluctuate according to the days of service or by spacing constraints between service days. These complications are treated in the two next subsections.

3.2 Varying demands - A simple classification

In general, service costs can be computed from the demands. For instance, waste management companies use approximate formulas to estimate the time required for collecting a street. This time is a function of the length of the street and of its total amount of waste.

Now, what about the demands? We call *class A* the set of problems, in which demands do not depend on a "production" that accumulates with time. The demand of each task (and its service cost) will be identical for any service day. One example is the pulverization of herbicides on rails to avoid weeds: the amount sprayed and the spraying time per meter are fixed.

The *class B* gathers problems like urban waste collection in which each task u has a production $prod(u,p)$ for each day p . Two special cases are a constant daily production $prod(u,p) = prod(u)$, and a global trend on the network, e.g. the peak of waste produced by lawn mowers after a hot and rainy period. Global trends can be modeled using a growth factor $\alpha(p)$ and a daily reference production $ref(u)$: the production for day p is then $prod(u,p) = ref(u) \cdot \alpha(p)$.

Class B problems may be *acyclic (subclass B1)* or *cyclic (subclass B2)*. In B1, the horizon is not cyclic and the demand $r(u)$ of task u in day p is the production accumulated since the previous service day. An example is when a new road shoulder mowing campaign starts in spring : the vegetation dies or stops growing in winter and the planning of last year can be forgotten. Let $s(1), s(2), \dots, s(f(u))$ the $f(u)$ successive service days selected for one task u . The demand $r(u,k)$ to be satisfied at the k -th service is then:

$$(2) \quad r(u, k) = \sum_{p=s(k-1)+1}^{s(k)} prod(u, p)$$

This formula is valid for the first visit ($k=1$) by assuming a fictitious visit at time 0. It can be amended to handle an initial stock.

In cyclic problems B2, H repeats itself but selected service days must be identical in all repetitions. In waste collection for instance, the amount collected for the first service in a week depends on the waste produced since the last service in the previous week. If the np daily productions of each arc are repeated each week, the task-to-day assignment can be determined for one single reference week. The demands are still computable by formula (2), except for $k=1$ for which days must be scanned from $s(f(u))+1$ to $s(1)$, cyclically.

Problems B2 are difficult to tackle by algorithms that build a PCARP solution day by day. For instance, Monday trips cannot be computed as long as the assignment is not known for all days (demands are still undefined).

3.3 Spacing constraints

Spacing constraints may be fixed at the strategic level and included in the data. For a task u , they can be defined either by a minimum spacing $spmin(u)$ and a maximum spacing $spmax(u)$ between two successive services, or by a set $comb(u)$ of $ncomb(u)$ day combinations like (1,2,..., np) for "everyday" or (1,3) for "Monday-Wednesday". Like in urban waste collection, we assume that all combinations in $comb(u)$ have the same frequency, *i.e.* they contain $f(u)$ days.

After several attempts to design GAs and linear programming models, we found the first system more difficult to handle, so we selected the day combinations. This choice is not restrictive because any pair ($spmin(u), spmax(u)$) can be converted into day combinations. Moreover, the conversion is not costly for small horizons like in waste collection. In Troyes for instance, $np = 7$, the streets are collected everyday (town center) or twice a week (suburbs), but never on the week-ends, and only three combinations are used among 11: (1,2,3,4,5), (1,4) and (2,5).

The day combination system has two big advantages: a) spacing constraints are satisfied implicitly, b) the demands in class B problems can be computed in advance for each day in each combination, using formula (2) or its adaptation for cyclic problems. After this data preparation, we obtain one single model for class A and class B problems.

4 A genetic algorithm for the PCARP

4.1 Relaxation of fleet size constraint

When only nva vehicles are available, an assignment of tasks to days is feasible if the tasks in each day can be partitioned into at most nva vehicle loads not greater than Q . This problem is already NP-complete since it reduces to a bin-packing problem for $np = 1$. In constrained cases, our first crossovers failed to build feasible children, even with a dedicated repairing procedure.

In fact, in response to a request for proposals issued by a city, waste management companies try to estimate the minimal resource investment (fleet size, main objective) before the minimal operating costs (total cost of trips possible with that fleet, secondary objective). We then decided to *relax the fleet size constraint* and to include it in the objective function.

Let S a solution costing $cost$ and using nvu vehicles, and M a constant larger than $cost$. If the fleet size is not imposed, our GA minimizes nvu in priority with a *hierarchical bi-objective function* $F_1(nvu, cost) = M \cdot nvu + cost$. If the fleet size is fixed to nva vehicles, the GA switches to $F_2(nvu, cost) = M \cdot \max(0, nvu - nva) + cost$. If even the GA cannot find solutions using nva vehicles, the fleet is certainly underestimated.

4.2 Chromosome structure

Recall that ns is the total number of services to be done over the horizon H . Any chromosome S for a PCARP solution is then a sequence of ns tasks, partitioned into np successive sub-lists (one per day). Each task u appears $f(u)$ times in total, as u or $inv(u)$, in $f(u)$ days corresponding to one allowed day combination contained in $comb(u)$. It appears at most once in each day. Note that the sub-list of tasks for one given day is similar to an ordinary CARP chromosome, as defined in 2.2, except that some tasks may be absent for that day.

The sequence can be viewed as a priority list for one single vehicle performing all tasks day by day. This coding is simple but pertinent, since at least one optimal sequence exists. Indeed, any PCARP solution can be defined as one set of trips per day, each trip being a list of tasks. Consider an optimal PCARP solution and concatenate all these lists into one single-sequence: this gives an optimal chromosome.

4.3 Chromosome evaluation

The idea is to evaluate the subsequence of tasks of each day with the splitting procedure described in 2.2 for the single-period CARP. Recall that this procedure indicates where to split the subsequence into trips at minimum cost. For the PCARP, it can be implemented as a procedure *split*(S, p, n, c), evaluating in S the subsequence for day p and returning the number of trips n for this day and their total cost c . For instance, the bi-objective functions F_1 of 4.1 can be computed as follows:

```

nvu := 0
cost := 0
for p := 1 to np
    split (S, p, n, c)
    nvu := max(nvu, n)
    cost := cost + c
endfor
F1 := M.nvu + cost.

```

In fact, the splitting procedure needs the demand associated with each task occurrence, to know which trips can be built. As mentioned at the end of 3.3, for any task u , the demands can be computed in advance for each day of each combination of $comb(u)$. If the day combination used for each task is kept in the chromosome, it is easy to design a function *amount*(S, k) called in *split* and returning the demand for any task occurrence $S(k)$.

Note that when the number of vehicles available nva is imposed (bi-objective function F_2), the total load per day may exceed $nva \cdot Q$ since the fleet size constraint is relaxed. Guided by F_2 , the GA will have the task of finding solutions using at most nva vehicles, if they exist.

4.4 Initial population

The population is a table *Pop* of nc chromosomes whose costs are always distinct. This avoids a premature convergence caused by the generation of identical solutions ("clones") and favors a better dispersal of solutions. We include in *Pop* two good solutions resulting from the constructive heuristics described in section 5. Their chromosomes are formed by concatenating their trips day by day.

Any other initial chromosome S is randomly generated as below. During the generation, S is coded as np sublists of tasks $L(1), L(2), \dots, L(np)$ (one per day). Note how feasibility is simply achieved thanks to the day combinations.

```

empty L(1), L(2), ..., L(np)
for count := 1 to nt
    draw one task u not yet assigned
    draw a day combination  $k \in comb(u)$ 
    for each day p of combination k
        add u at the end of L(p)
    endfor
endfor
concatenate L(1), ..., L(np) into S
evaluate S

```

4.5 Crossover

This is the most complicated component of the genetic algorithm for the PCARP. The classical LOX crossover (Linear Order Crossover) is well known for permutation chromosomes. It partly transmits the order of elements from the parents to their children. Consider two parents P1 and P2 of n elements (permutations of the integers 1 to n). The construction of the first child C1 by LOX is as follows (the other child is obtained by exchanging the roles of P1 and P2):

```

draw a and b with  $1 \leq a \leq b \leq np$ 
copy P1(a)...P1(b) into C1(a)...C1(b)
j := 0
for i := 1 to n
    if P2(i) not already in C1 then
        j := j+1
        if j = a then j := b+1 endif
        C1(j) := P2(i)
    endif
endfor.

```

So, LOX draws a substring in P1 and copies it at the same locations into C1. Then P2 is scanned from left to right. Its elements not yet present in C1 are copied to fill the empty positions of C1, from left to right too.

Our crossover for the PCARP, ELOX, extends LOX to chromosomes divided into days and in which the tasks can now occur more than once. It uses an auxiliary function *day*(S, k) returning the day corresponding to a task occurrence $S(k)$.

Like for the random generation in 4.4, the child C1 is prepared as a table of np sublists C1(1), C1(2), ..., C1(np), one per day. ELOX starts by randomly drawing two cutting sites a, b in P1. The tasks in P1(a) to P1(b) are then copied into C1, while keeping their service day. P2 is then scanned to complete C1. Contrary to LOX, $f(u)$ occurrences of each task u must be put into C1 to satisfy frequencies. An array *numb* stores the number of occurrences *numb*(u) already copied

for each task u . The set of days already selected for u in $C1$ is called $done(u)$.

ELOX copies a task u from $P2(k)$ only if its frequency is not yet satisfied. It tries first to keep the service day p of u . This may not be possible if $C1$ already contains a copy of the task for that day, or if no day combination for u includes $done(u) \cup \{p\}$. In that case, ELOX uses the earliest day p not containing u and such that $done(u) \cup \{p\}$ is included in at least one day combination of $comb(u)$.

It is easy to see that p always exists. ELOX really is a generalization of LOX since it reduces to LOX in the single-period case ($np=1$, $f(u)=1$ for all u). The whole procedure is detailed in Algorithm 1, with an example in Figure 2. Let $comb(uk)$ the k^{th} day combination in $comb(u)$. The function $dayOK(p,u)$ is true iff a task u from $P2$ can be kept in day p , i.e. if $p \notin done(u)$ and $\exists k$ in $1..ncomb(u)$ with $done(u) \cup \{p\} \subseteq comb(uk)$.

```

prepare np empty daily lists C1(1),C1(2),...,C1(np)
for each task u; numb(u):=0; done(u)=∅; endfor
draw randomly two cutting sites a and b in P1
for k := a to b //copy P1(a)..P1(b) into C1 (keep sequence and days)
    u := P1(k); p := day(P1,k)
    copy u at the end of C1(p)
    increment numb(u) (and numb(inv(u)) for an edge-task)
    add p to done(u) (and to done(inv(u)) for an edge-task)
endfor
for k := 1 to ns //scan P2 to finish C1
    u := P2(k); p := day(P2,k)
    if numb(u) < f(u) then //if frequency not yet satisfied
        if not dayOK(p,u) then //day must be changed
            p := earliest day such that dayOK(p,u)
        endif
        copy u at the end of C1(p);
        increment numb(u) (and numb(inv(u)) for an edge-task)
        add p to done(u) (and to done(inv(u)) for an edge-task)
    endif
endfor.

```

Algorithm 1: ELOX crossover for the PCARP

5 Computational evaluation

5.1 Benchmark instances

We have built 23 PCARP files from 23 CARP instances with 11 to 51 tasks, used for example in [6]. Each edge u has $c(u)=w(u)$. To imitate the situation in Troyes, we add a cyclic horizon of 7 days, with a fleet idle on days 67. For each edge u , a frequency $f(u)$ in $1..5$ and a set of day combinations $comb(u)$ are randomly selected.

4.6 Other GA ingredients

The *incremental replacement* is used. At each iteration, two parents $P1$ and $P2$ are randomly selected in Pop by a *binary tournament method*: two chromosomes are randomly selected and the best becomes $P1$, the same procedure is repeated to get $P2$. ELOX is applied to $P1$ and $P2$ to generate two children $C1$ and $C2$. One child is discarded at random. The remaining child may undergo mutation, in fact a local search already used in [6] for the basic CARP. It improves the trips in each day by iteratively moving a task or swapping two tasks. If a duplicate cost occurs, the child is rejected. If not, it finally replaces in Pop a chromosome drawn above the median cost. The population of distinct solutions must be relatively small ($nc=30$ to 40) to avoid excessive rejection rates.

The GA stops after a maximum number of iterations or a maximum number of iterations without improving the best solution.

For each edge u and each day p in each combination, the demand in the CARP is viewed as a constant daily production and the PCARP demand $r(u,p)$ is computed with formula 2 of 3.2. The traversal cost $c(u)$ is kept. The service cost $w(u,p)$ is equal to $c(u)$ times the number of days since the previous day in the day combination considered. The resulting instances have a chromosome length ns ranging from 33 to 132 (average 70).

Task u	$f(u)$	$comb(u)$
1	3	{Mon,Wed,Thu}, {Tue, Wed,Thu}
2	2	{Mon,Wed}, {Tue,Thu}
3	1	{Mon}, {Thu}
4	1	{Mon}
5	1	{Mon}, {Tue}
6	3	{Tue,Wed,Thu}, { Mon,Tue,Thu}, { Mon,Tue,Wed}
7	2	{Tue,Thu}, {Mon,Wed}
8	1	{Tue}, {Thu}

	Monday				Tuesday			Wednesday			Thursday			
P1	1	2	3	4	<u>5</u>	<u>6</u>	<u>7</u>	<u>1</u>	2	6	1	6	7	8
P2	7	6	5	4	8	6	2	1	7	1	3	6	1	2
C1	6	4	<u>5</u>	<u>6</u>	<u>7</u>	8	2	1	<u>1</u>	6	7	3	1	2

Figure 2: An example of two parents with one child C1 generated by the ELOX crossover (the tasks copied from P1 are underlined and in boldface)

5.2 Algorithms compared

Four algorithms are evaluated for their ability to minimize the bi-objective function presented in 4.1, $F_1(nvu, cost) = M.nvu + cost$. The first one is a best insertion heuristic (BIH) used as a basis for comparison. BIH starts with empty trips in each day and a list of required edges sorted in decreasing order of frequencies. Starting from the first task, each iteration inserts the current task u into $f(u)$ distinct days in order to minimize F_1 . This is achieved by testing each possible day combination of u , each day p of the combination and the following insertions of u into day p : all feasible insertions (in terms of capacity) into existing trips and the creation of a new trip reduced to task u . BIH is expected to give reasonably good solutions, close to the ones obtained by a human planner.

The other algorithms use genetic algorithms on the whole horizon or on individual days. Instead of using the new GA to work on the horizon and

the existing GA of [6] (recalled in 2.2) for a given day, we prefer to use one single GA, the new one. This is possible because the new GA behaves like the GA already published for the CARP when it is executed with $np=1$.

The second algorithm proceeds in two steps like many methods used for production planning. In the first step, the tasks are assigned to days (tactical decision level). This is done by running BIH and by keeping only the assignment of tasks to days (the trips are discarded). In the second step, the tasks assigned to each day are formatted as single-period CARP problems. These CARP are solved by calling the new GA with $np=1$ (operational level). This technique is expected to give excellent solutions for each CARPs, but the total cost depends on the quality of the assignment, which cannot be modified. We call this method DGA (for daily GA).

The third method runs the new GA that considers simultaneously tactical and operational level decisions: this algorithm can modify

at each crossover the assignment of tasks to days and the trips in each day. This method is expected to give better results, because of its wider neighborhood and its ability to improve a weak initial assignment. We call it PGA (like Periodic GA).

The fourth and last method consists in a post-optimization of the results found by PGA. Like for DGA, we keep the assignment of tasks to days and forget the trips. A single-period CARP is defined from the tasks in each day. The new GA is called to solve these CARPs. This method called IGA (improved GA) can be viewed as a

final intensification of DGA, because the population is concentrated on individual days. It is expected to give the best results, but at the expense of a larger CPU time.

5.3 Results

All algorithms are programmed in Delphi 5 and executed on a 1 GHz Pentium III PC with Windows 98. All GAs work with a population of 30 chromosomes and perform 40000 crossovers. Each child undergoes a local search with rate 0.2. The average results are shown in Table 1.

Table 1: Comparison of BIH, DGA, PGA and IGA on 23 PCARP instances

AVERAGE VALUES	BIH	DGA	PGA	IGA
Fleet size <i>n_{vu}</i>	7.00	4.04	3.65	3.56
Cost of trips <i>cost</i>	968.4	689.7	706.3	688.9
CPU time per instance (min)	< 1 s	4.63	1.09	2.12

These results confirm the predictions. The insertion heuristic gives disappointing results that can be explained by the insertion of each task in several days: for instance, for a combination of two days, an excellent insertion in one day can be cancelled by a costly insertion in the other day. The table confirms the interest of tackling tactical and operational decisions in parallel. Even without post-optimization, PGA is better than DGA in terms of vehicles, although the total cost of the trips is increased. The post-optimized version IGA gives the best results, both in terms of vehicles and total cost.

6 Concluding remarks

We have defined a new problem in arc routing, the PCARP, and developed a GA addressing simultaneously the tactical and operational decision levels. The testing on 23 instances shows the interest of this technique compared to more classical two-phase methods. As mentioned in introduction, our algorithms are not restricted to the undirected PCARP: they can tackle extensions like prohibited turns, mixed graphs, etc. These promising results show the interest of integrating tactical and operational decisions in long-term planning problems raised by many applications of arc routing.

References

- [1] Belenguer, J.M. and Benavent, E. (1997). *A cutting plane algorithm for the capacitated arc routing problem*, Research Report, Dept. of Statistics & OR, Univ. of Valencia, Spain.
- [2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduction to algorithms*, The MIT Press, 1990.
- [3] B.L. Golden, J.S. DeArmon, E.K. Baker, Computational experiments with algorithms for a class of routing problems, *Computers & Operations Research*, 10(1), 47-59, 1983.
- [4] B.L. Golden, R.T. Wong, Capacitated arc routing problems, *Networks*, 11, 305-315, 1981.
- [5] A. Hertz, G. Laporte, M. Mittaz, A tabu search heuristic for the capacitated arc routing problem, *Operations Research*, 48(1), 129-135, 2000.
- [6] P. Lacomme, C. Prins, W. Ramdane-Chérif, A genetic algorithm for the capacitated arc routing problem and its extensions, 473-483, in *Applications of evolutionary computing*, E.J.W. Boers (ed.), Lecture Notes in Computer Sciences 2037, Springer, 2001.