Multiobjective Capacitated Arc Routing Problem

Conference Paper *in* Lecture Notes in Computer Science · April 2003

DOI: 10.1007/3-540-36970-8_39 · Source: DBLP

3 authors:

Philippe Lacomme
Université Clermont Auvergne
168 PUBLICATIONS **2,790** CITATIONS

SEE PROFILE

Christian Prins
Université de Technologie de Troyes
121 PUBLICATIONS **5,640** CITATIONS

SEE PROFILE

Marc Sevaux
Université Bretagne Sud
297 PUBLICATIONS **3,195** CITATIONS

SEE PROFILE

Multiobjective Capacitated Arc Routing Problem

P. Lacomme[1], C. Prins[2], and M. Sevaux[3]

[1] Université Blaise Pascal
CNRS, UMR 6158 - LIMOS/Laboratoire d'Informatique
Campus Universitaire des Cézeaux - BP 10125 - F-63177 Aubière Cedex, France
lacomme@isima.fr
[2] Université de Technologie de Troyes
Laboratoire d'Optimisation des Systèmes Industriels
12 Rue Marie Curie - BP 2060 - F-10010 Troyes Cedex - France
prins@utt.fr
[3] Université de Valenciennes
CNRS, UMR 8530 - LAMIH/Systèmes de Production
Le Mont Houy - F-59313 Valenciennes Cedex 9 - France
msevaux@univ-valenciennes.fr

Abstract. The Capacitated Arc Routing Problem (CARP) is a very hard vehicle routing problem raised for instance by urban waste collection. In addition to the total route length (the only criterion minimized in the academic problem), waste management companies seek to minimize also the length of the longest trip. In this paper, a bi-objective genetic algorithm is presented for this more realistic CARP, never studied before in literature. Based on the NSGA-II template, it includes two-key features: use of good constructive heuristics to seed the initial population and hybridization with a powerful local search procedure. This genetic algorithm is appraised on 23 classical CARP instances, with excellent results. For instance, for a majority of instances, its efficient solutions include an optimal solution to the academic CARP (minimization of the total route length).

1 Introduction: the CARP and its Applications

The Capacitated Arc Routing Problem (CARP) is defined in the literature on an undirected graph $G = (V, A)$ with a set V of n nodes and a set A of m edges. A fleet of identical vehicles of capacity W is based at a depot node s. A subset R of edges require service by a vehicle. All edges can be traversed any number of times. Each edge (i, j) has a traversal cost c_{ij} and a demand q_{ij}. The goal is to determine a set of vehicle trips (routes) 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 handled by any vehicle does not exceed W.

Since the CARP is $\mathcal{NP}$-hard, large scale instances must be solved in practice with heuristics. Among fast constructive methods, one can cite for instance

Path-Scanning [10], Augment-Merge [11] and Ulusoy's splitting technique [19]. Metaheuristics available include tabu search methods [2, 12] and the memetic algorithm of Lacomme, Prins and Ramdane-Chérif [14], which is currently the most efficient solution method. All these heuristic algorithms can be evaluated thanks to very good lower bounds [1, 3].

CARP problems are raised by operations on street networks, e.g. urban waste collection, snow plowing, sweeping, gritting, etc. Economically speaking, the most important application certainly is municipal refuse collection. In that case, nodes in G correspond to crossroads, while edges in A model street segments linking crossroads. The demands are amounts of waste to be collected. The cost on each edge is generally a time.

The academic CARP deals with only one objective, minimizing the total duration of the trips. In fact, waste management companies are also interested in balancing the trips. For instance, in Troyes (France), all trucks leave the depot at 6 a.m. and waste collection must be finished as soon as possible to assign the crews to other tasks, e.g. sorting the waste at a recycling facility. So, the company wishes to solve in fact a bi-objective version of the CARP, in which both the total duration of the trips and the duration of the longest trip (*makespan*) are to be minimized.

In the last decade, there has been a growing interest in multi-objective vehicle routing problems, but all published papers deal with node routing problems, *i.e.*, in which the tasks to be performed are located on the nodes of a network [16, 5, 13, 18]. This paper is the first study devoted to an arc routing problem with multiple objectives.

The remainder of the paper is organized as follows. Section 2 recalls some definitions on multi-objective optimisation. Section 3 introduces the NSGA-II template and our specific components, while numerical experiments are conducted in Section 4. Conclusion is drawn in Section 5.

2 Multi-objective Optimisation

The growing interest in multi-objective optimisation results from the numerous industrial applications where solving a single objective optimisation problem is an abusive simplification. In such cases, computing an optimal solution for one objective results in poor values for other criteria. Hence, developing some algorithms giving a compromise between several criteria is better suited to industrial reality.

Multi-objective problems in combinatorial optimization are often very hard to solve: even their single-objective versions are generally $\mathcal{NP}$-hard. This is why lots of papers have been recently published on evolutionary techniques that are able to handle several objectives at a time and give efficient solutions by the mean of an evolving population.

For an extensive information on multi-objective optimisation see [4]. A recent survey and annotated bibliography [9] can help for finding the best suitable method to tackle a multi-objective optimisation problem (see also [6, 7]).

A multi-objective optimisation problem (MOOP) can be formulated as follows :

$$\begin{array}{ll} \text{minimize} & \{f_1(x), f_2(x), \ldots, f_{\text{nc}}(x)\} \\ \text{subject to} & x \in S \end{array} \quad (1)$$

where nc is the number of criteria , $x = (x_1, x_2, \ldots, x_{\text{nv}})^T$ a vector of nv decision variables and S a set of feasible solutions.

Definition 1. *A solution x **dominates** a solution y, $x \succ y$ if*
x is at least as good as y for each objective: $\forall k \in [1, \ldots, nc], f_k(x) \leq f_k(y)$ and
x is better than y for at least one citerion: $\exists k \in [1, \ldots, nc], f_k(x) < f_k(y)$.

According to this definition, x is *non-dominated* if there is no solution y that dominates x. The set P of all non-dominated solution is called *non-dominated set* or *efficient set (E)* or *pareto-optimal set/front*. And for a set of solutions $U \supset P$, $\forall x \in P, \forall y \in P, x \not\succ y, y \not\succ x$, and $\forall y \in U \setminus P, \exists x \in P, x \succ y$. Hence a criterion of a solution in the non-dominated set cannot be improved without detariorating another criterion.

3 NSGA-II Implementation

Today, many multi-objective optimisation algorithms are available and selecting the best one for a given application is not obvious. For an up-to-date description and evaluation of these algorithms the reader can refer to [20]. Nevertheless, this survey evaluates the algorithms on a specific set of problems and it is difficult to generalize the conclusions to other problems.

Our choice then turned to one of the most widely used multi-objective genetic algorithm (MOGA), NSGA-II [8]. This MOGA has been adapted to match our requirements concerning the bi-objective CARP. Minor bugs from [6, 7] were also corrected. The bug-free algorithms are provided in the next section.

3.1 NSGA-II for two objectives

The NSGA-II algorithm is based on the notion of non-dominated fronts. It improves an earlier algorithm called NSGA in terms of complexity for calculating the non-dominated fronts. Each iteration of NSGA-II starts by sorting the current set of solutions U by non-domination. This step calculates the non-dominated subset of U, remove its solutions from U and repeats this process until U becomes empty.

The set U is in practice a table or a list of solutions. $U(i)$ denotes the i^{th} solution of U. To save space, the fronts store solution indexes, not the solutions themselves. Dominance between two solutions x and y is computed by the $\text{dom}(x, y)$ boolean function. `nb_better(i)` counts the solutions which dominates the solution $U(i)$, `set_worse(i)` is the set of solutions dominated by

$U(i)$. The non-dominated sorting partitions the solutions into nf fronts denoted $\text{front}(1), \text{front}(2), \ldots, \text{front}(\text{nf})$. `rank(i)` denotes the index of the front that stores solution $U(i)$.

The first step of the sorting process computes in $O(nc.ns^2)$ `nb_better` and `set_worse` values, the second phase in $O(ns^2)$ assigns solutions to their respective fronts and computes nf.

```
procedure non-dominated-sort()
  front(1) := emptyset
  for i := 1 to ns do
    nb_better(i) := 0
    set_worse(i) := emptyset
    for j := 1 to ns do
      if dom(U(i),U(j)) then add j to set_worse(i)
      elseif dom(U(j),U(i)) then nb_better(i) = nb_better(j)+1 endif
    endfor
    if nb_better(i) = 0 then add i to front(1) endif
  endfor
  nf := 1
  loop
    for each i in front(nf) do
      rank(i) := nf
      for each j in set_worse(i) do
        nb_better(j) = nb_better(j)-1
        if nb_better(j) = 0 then add j to front(nf+1) endif
      endfor
    endfor
    exit when front(nf+1) = emptyset
    nf = nf+1
  endloop
```

Comparing solutions in NSGA-II is done by two means. If two solutions belongs to two differents fronts, the smaller rank gives the better solution. If two solutions belong to the same front, the algorithm privileges the most isolated one by computing a crowding distance. The idea is to favor the best solutions while keeping the fronts well scattered to prevent clusters of solutions.

Let call *margin* the crowding distance function. The best solution among two candidates can then be computed thanks to a boolean function `better(i,j)`, for which a solution $U(i)$ is better than a solution $U(j)$ if (`rank(i) < rank(j)`) or (`rank(i) = rank(j)` and $\text{margin}(i) > \text{margin}(j)$).

For a front R of nr solutions and $\text{nc} = 2$ criteria, margins are calculated as follows. R is sorted in increasing values of the first criteria. Let $R(k)$ the k^{th} solution of the sorted front. If c is one of the criteria, let f_c^{kmin} and f_c^{kmax} the minimum and maximum values of f_c in R. For $k = 1$ or $k = \text{nr}$, let $\text{margin}(k) = \infty$ to be sure that these two extreme points will be chosen if necessary. For $1 < k < \text{nr}$,

$$\text{margin}(k) = \frac{f_1(R(k)+1) - f_1(R(k)-1)}{f_1^{max} - f_1^{min}} + \frac{f_2(R(k)-1) - f_2(R(k)+1)}{f_2^{max} - f_2^{min}} \quad (2)$$

The overall structure of NSGA-II is given below. The population, denoted by `pop`, is composed of ns chromosome records. Each record includes also the two criteria values, the rank and the margin of the individual. The procedure `first_pop(pop,ns)` builds the initial population.

The procedure `non_dominated_sort(pop,ns,front,nf)` sorts `pop` by non-domination with the algorithm previously defined. It returns the number of different fronts nf and the fronts themselves (front(i) is the i^{th} front and front(i,j) is the index of the j^{th} solution of front(i)). This procedure computes also the rank of each solution in `pop`, *i.e.* the index of the front the solution belongs to.

The procedure `add_first_children(pop,ns)` creates ns offsprings which are added at the end of `pop`, thus doubling its size. Parents are selected by binary tournament.

```
procedure nsga2()
  first_pop(pop,ns)
  non_dominated_sort(pop,ns,front,nf)
  add_first_children(pop,ns)
  repeat
    non_dominated_sort(pop,ns,front,nf)
    for i := 1 to nf do
      get_margins(front,i,pop)
      margin_sort(front,i,pop)
    endfor
    newpop := emptyset
    i := 1
    while |newpop|+|front(i)| <= ns do
      add front(i) to newpop
      i := i+1
    endwhile
    missing := ns-|newpop|
    for j := 1 to missing do
      add front(i,j) to newpop
    endfor
    pop := newpop
    add_children(pop,ns)
  until (stopping_criterion)
```

Each iteration of the *repeat* loop consists of keeping the ns best solutions and adding the ns offsprings of the next generation. The first *for* loop computes the margins for each front i with the `get_margins(front,i,pop)` procedure, then, in each front, solution are sorted in decreasing margin order by the `margin_sort(front,i,pop)` procedure.

The ns best solutions of `pop` are copied into `newpop` front by front until the new population is filled. Finally, `newpop` replaces `pop`.

3.2 Components for the GA

This section describes the components used in the NSGA-II algorithm to solve the multi-objective CARP. Some of them were initially developped for a GA that

is currently the most effective solution method for the single-objective CARP [14]. Only a short description is recalled here.

Chromosome Representation and Evaluation Following [14], a chromosome σ is a permutation of the required arcs, without trip delimiters. It can be seen as a giant tour ignoring vehicle capacity. Implicit shortest paths are assumed between the depot and the first task, between two consecutive tasks, and between the last task and the depot.

Trip delimiters are not necessary because a procedure described in [14] is able to derive a least total cost CARP solution by splitting the chromosome (giant tour) into capacity-feasible tours. The procedure is based on the computation of a shortest path in an auxiliary graph, in which each arc corresponds to one sub-sequence of tasks (one feasible tour for the CARP) that can be extracted from the giant tour.

Note that the use of a splitting procedure to define trip limits allows a simple encoding of chromosomes and the use of traditional crossover operators designed for permutation chromosomes, e.g. OX or LOX.

Initial Population Like in other MOGAs, an initial population has to be generated. Generating permutations randomly is a classical technique, but adding high-quality solutions always accelerates the convergence of the algorithm.

Several good solutions are added to the initial population. They are computed by extending three classical heuristics for the CARP, namely Path-Scanning [10], Augment-merge [11] and Ulusoy's heuristic [19]. These solutions will be denoted by PS, AM and UL and will be used in the numerical experiments (see details in Section 4.1).

Crossover Operators For the single objective CARP, the standard LOX and OX crossover operators give the best results and following the conclusions of [14], the OX crossover operator has been selected for the multi-objective problem. Details on its implementation can be found in [14].

Stopping Conditions Today, a major problem is raised when developing a MOGA. Among the different methods, there is no standard technique to define reliable stopping conditions. In combinatorial optimisation and in the single-objective case, metaheuristics are often stopped after a fixed number of iteration without improving the current best solution. Since it is difficult to extend this method to multi-objective optimisation, authors of published MOGAs generally stop their algorithms after a fixed number of iterations. For this study, we will do the same, although we are currently investigating more sophisticated stopping criteria.

3.3 Local Search Procedures

In single-objective optimization, it is well known that a standard GA must be hybridized with local search (LS) operators to become competitive [15] and this property is also true in multi-objective optimization. Of course, several adaptations are required to convert a local search procedure for the single-objective case into an efficient bi-objective local search procedure.

Without going into details, we apply a local search with a fixed probability (called the *rate* of the local search) to each offspring resulting from a crossover. The child chromosome is first converted into a valid CARP solution (using the splitting procedure) and five types of moves are examined for each pair of required edges (tasks): 1) inverse the traversal direction of a task in its trip, 2) move one task after one other, 3) move two adjacent tasks after one other, 4) swap two tasks and 5) perform 2-opt moves. More information on the single-objective local search procedure is given in [14]. A descent algorithm is used with these neighborhoods.

A descent algorithm with the same neighborhoods is also used for the bi-objective CARP. The main change resides in the conditions for accepting a move. The variations of each objective function are computed. `CVar` denotes the total cost variation and `MVar` the makespan variation. Three acceptance criteria are defined:

Accept1 is `True` if `CVar <= 0`.
Accept2 is `True` if `MVar <= 0`.
Accept3 is `True` if `(CVar<=0 and MVar<0) or (CVar<0 and MVar<=0)`.

The last version, Accept3, leads to a true bi-objective local search, in which a move is accepted if and only if its result dominates the current solution. Accept1 and Accept2 are two single-objective versions that improve either the total cost or the makespan, but not both.

4 Computational Experiments

All developements are done in Borland ©Delphi 6 and tested on a Pentium Celeron 650 Mhz with Windows 98. The computational evaluation uses the standard benchmark problems from Golden, DeArmon and Baker [10] which are used by all authors working on the single-objective CARP. Files can be downloaded from http://www.uv.es/ belengue/carp.html.

This set of instances is composed of 25 files from 7 to 27 nodes and 11 to 55 edges. In fact, two instances (originally gdb8 and 9) are inconsistent and never used. The remaining instances are re-numbered gdb1 to gdb23. All edges are required.

4.1 Test protocol and parameters

A first execution of the MOGA has been done and the results are used to provide a reference front for the other test protocols. This first run is done without the

local search procedure but with the initial heuristics PS, AM and UL presented in Section 3.2. This reference experiment will be denoted by MO1. Table 1 lists the test protocols and parameters used for the other numerical experiments. The protocols are numbered from MO1 to MO6, the population size is always fixed to 60 individuals and the crossover operator is OX. The maximum number of iterations is 100, except for the test MO5 where no local search and no initial heuristics are used. The last two columns specify the type of local search procedure used (corresponding to the "Accept" function in the LS procedure) and the LS rate.

Table 1. Test protocol and parameters used

Test	Pop.	XOver	Max	Heuristics			Local Search	
Protocol	size	op.	# it.	PS	AM	UL	LS	%LS
MO1	60	OX	100	Yes	Yes	Yes	No	–
MO2	60	OX	100	Yes	Yes	Yes	1	10
MO3	60	OX	100	Yes	Yes	Yes	2	10
MO4	60	OX	100	Yes	Yes	Yes	3	10
MO5	60	OX	300	No	No	No	No	–
MO6	60	OX	100	Yes	Yes	Yes	3	20

A MOGA with 60 individuals and 100 iterations performs a total of 6000 crossover operations. On a specific instance, the evolution of the different number of iterations will be shown.

4.2 Measuring the deviation from a reference front

Comparing two different methods for solving the same multi-objective problem is very difficult, especially when no information is known concerning the final choices of the decider. Nevertheless, this step cannot be avoided in this study, at least to decide which test protocol is best suited for solving our problem.

A measure, presented in [17], has been enhanced and applied to compare each of the test protocol with the reference protocol MO1. The front obtained by MO1 will be considered as the reference front. Given a reference front, a kind of distance is measured between each solution to be compared and its projection onto the extrapolated reference front. The extrapolated reference front is a piecewise continuous line (for the bi-objective case) going through each solution of the front and extended beyond the two extreme solutions. Figure 1 gives an example of the extrapolated front and such a measure.

For the example of Figure 1, two measures can be given, the simplest is: Measure $= d1 + d2 + d3 + d4$, but in some cases, it could be interesting to normalize this measure by dividing it by the number of efficient solutions in front E to compare: Norma = Measure $/|E|$. If one of the solution of the front to compare is under the (extrapolated) reference front, the distance will be of course negative.

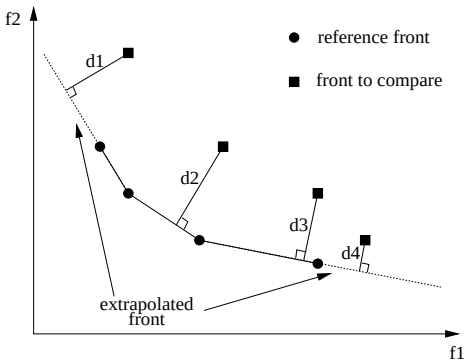

Fig. 1. Example of the distance to the extrapolated front

This measure is a bit arbitrary and of course subject to criticism. For example, what can be the measure if the reference front is reduced to only one solution? How can we measure the difference between two fronts where one is well spread while the other presents several "holes" between clusters of solutions? Nevertheless, its merit is to exist and to help us to rank the tested protocols.

Because of no a-priori knowledge of the solutions which lie on the efficient front, this measure should be completed by a visual inspection.

4.3 Lower Bounds and Heuristics

For the two criteria, lower bounds and heuristic results are presented in Table2. Column "File" gives the filename of the instance and columns "n" and "τ" give the number of nodes and the number of required edges for each instance. Columns "LBC" and "LBM" denote the lower bounds for the total cost and the makespan. Columns "PSC" and "PSM" report the total cost and the makespan given by the Path-Scanning heuristic (AM stand for Augment-Merge heuristics and UL for Ulusoy's heuristic, the last letter C - resp. M - is used for the total cost - resp. makespan). See Section 3.2 for more details.

The lower bound LBC for the total cost is given by Belenguer and Benavent [2] except for gdb14 given by Amberg and Voß [1]. The lower bound LBM for the makespan is defined by Lacomme *et al.* [14].

Best-known solutions for the single objective (total cost) optimization are not reported because they are equal to the LBC lower bound for 21 files out of 23. The memetic algorithm developped in [14] gives the 21 best solutions and all are equal to their lower bounds.

Note that the heuristics already find four least-cost solutions (files gdb4, gdb7, gdb15 and gdb17) and two solutions of minimal makespan (files gdb13 and gdb19). These solutions are used to provide the NSGA-II algorithm with very good starting points.

Table 2. Lower bounds and heuristic results for the *gdb* files (*optimal solution for one criterion)

			Lower Bounds		Heuristics					
File	n	τ	LBC	LBM	PSC	PSM	AMC	AMM	ULC	ULM
gdb1	12	22	316	63	350	91	349	91	330	107
gdb2	12	26	339	59	366	79	370	73	353	84
gdb3	12	22	275	59	293	79	319	71	297	84
gdb4	11	19	287	64	287*	91	302	104	320	91
gdb5	13	26	377	64	438	108	423	88	407	130
gdb6	12	22	298	64	324	87	340	84	318	102
gdb7	12	22	325	57	363	87	325*	74	330	91
gdb8	27	46	344	38	463	73	393	41	388	53
gdb9	27	51	303	37	354	57	352	44	358	53
gdb10	12	25	275	39	295	84	300	89	283	81
gdb11	22	45	395	43	447	114	449	97	413	121
gdb12	13	23	448	93	581	122	569	103	537	99
gdb13	10	28	536	128	563	203	560	128*	552	190
gdb14	7	21	100	15	114	26	102	28	104	37
gdb15	7	21	58	8	60	21	60	16	58*	27
gdb16	8	28	127	14	135	40	129	30	132	40
gdb17	8	28	91	9	93	31	91*	20	93	23
gdb18	9	36	164	19	177	44	174	42	172	41
gdb19	8	11	55	17	57	24	63	17*	63	25
gdb20	11	22	121	20	132	46	129	39	125	33
gdb21	11	33	156	15	176	44	163	34	162	48
gdb22	11	44	200	12	208	32	204	35	207	48
gdb23	11	55	233	13	251	50	237	31	239	52

As mentioned in [14], the lower bound LBM computed for the makespan criteria is not tight and the deviation to this bound can be large. Results of the next section will emphasize this point.

4.4 Numerical Evaluation for the Test Protocols

To study the efficiency of the method through each test protocol and without any information on the optimal efficient front, we have based our analysis on a deviation from the known lower bounds and also on the measure defined in Section 4.2. Average results are reported for the 23 *gdb* files.

Table 3 presents the results for each test protocol. Deviations (in %) from the lower bounds are reported for the extreme points of the approximate efficient front (these points are named LEFT and RIGHT). Again, C denotes the total cost objective and M the makespan objective. For LEFTC (the minimum values obtained for the total cost objective) average deviations are reported but also the number of optimal solutions found (when the lower bound is reached). MO1 is the reference front used for the measure. Two columns report the measure in its standard and normalized forms defined in Section 4.2.

For instance, in the test protocol MO4, the average number of efficient solutions on the last front is 3.26 (average among the 23 instances files); the average deviation from the total cost lower bound is only 0.56% and 16 out of 23 optimal solutions are retrieved. The average deviation from the makespan lower bound of the best solution for the total cost criterion is 46.35%. The average deviation from the makespan lower bound is 22.07% for a deviation of the total cost criterion of 9.72%. 11 out of 23 optimal solutions were retrieved for the makespan

criterion. Using our measure, the improvement of the reference front is indexed by -22.82 and -6.48 if this measure is normalized (divided by the number of solution in the efficient front).

Table 3. Results of each test protocol

Test	Aver.	LEFTC		LEFTM	RIGHTC	RIGHTM		Measure		CPU
Protocol	$\|E\|$	dev.	# opt.	dev.	dev.	dev.	# opt.	Std.	Norm.	time (s)
MO1	3.87	3.81	5	77.02	15.44	29.64	9	–	–	7.14
MO2	3.52	0.47	17	57.58	12.70	30.43	7	-18.15	-4.62	11.39
MO3	3.78	3.40	5	66.85	15.14	24.97	11	-5.56	-1.74	8.61
MO4	3.26	0.56	16	46.35	9.72	22.07	11	-22.82	-6.48	12.64
MO5	4.35	10.15	2	71.14	20.42	31.98	0	+35.41	+7.36	12.59
MO6	3.61	0.36	16	47.87	10.53	19.57	10	-26.17	-6.47	22.68

Looking at the measure columns, it appears that MO4 and MO6 test protocols better improve the reference front given by MO1. These two test protocols use the bi-objective LS procedure with the Accept3 function.

Less significant improvements are also obtained by test protocols MO2 and MO3. Local search is applied in one direction only (one criterion) and overall best solutions cannot be reached. As previously mentioned, the makespan lower bound is weak and gaps are rather large on average (from about 20% to 32%, column RIGHTM, dev.). However, some optimal solutions are found for this objective. The LBM bound corresponds in fact to the trivial solution in which each required edge is treated by a separate trip. A careful analysis of successive fronts show that this very particular solution is very unstable and quickly disappears. This is not very important because it is unlikely to be used in practice, due to its excessive number of vehicles.

The use of initial heuristics (as in the single criteria optimization) is very effective, and results of MO5 test protocol without these heuristics are far from the reference front, even with a larger number of iterations.

Increasing the local search rate from 10% to 20% while keeping the same number of iterations (test protocols MO4 and MO6) gives better solutions in terms of deviations to the lower bounds for the extreme points, at the expense of a very small increase for LEFTM and RIGHTC. The standard measure is also improved but, since the number of solutions in the efficient front is increased, the normalized measure is almost identical. Another price to pay for the higher LS rate is an increased average running time.

4.5 Influence of the Local Search Procedure

To highlight the importance of the use of the local search procedure, the second but largest instance *gdb11* is intensively tested. Two executions are compared: 1) No local search is applied, and 2) Systematic local search with rate 100%. Every 100 iterations the solutions of the approximate efficient front are scanned. The execution is stopped when two fronts (separated by 100 iterations) are identical.

Figure 2 presents the efficient solutions computed by the MOGA with and without local search procedure, after 100 iterations and at the end. 900 iterations are performed for the run without local search and 700 for the run with local search.

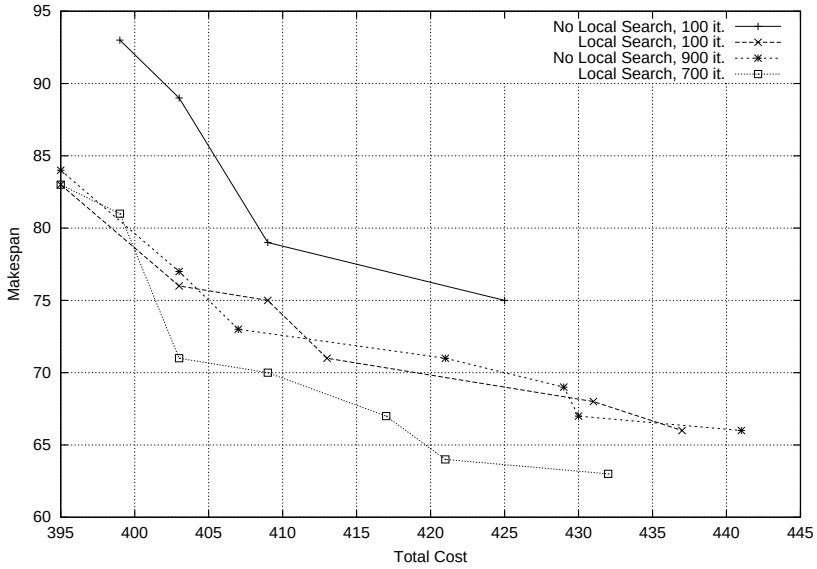

Fig. 2. Impact of local search on the efficient solutions for instance *gdb11*

It clearly appears that the LS procedure speeds up the convergence of the algorithm. The final solutions without local search are quite equivalent to the ones obtained after only 100 iterations with local search. If we compare the front obtained after 100 iterations when the local search procedure is activated and the final front whitout local search procedure, only two solutions of the former front are dominated by the solution of the latter front after 700 iterations.

Letting the algorithm with local search procedure running until the stopping conditions are met improves again the efficient front and provides high-quality results.

4.6 Evolution of the Efficient Fronts

Figure 3 shows the persistence of a solution in the efficient fronts during the iterations. Again, instance *gdb11* without local search procedure has been used here. As in Section 4.5, the algorithm is scanned every 100 iterations and stopped when no new solutions are found on the efficient front.

A grey box indicates that a solution has appeared in the efficient front. A hatched box indicates that the same solution 100 iterations before still belongs

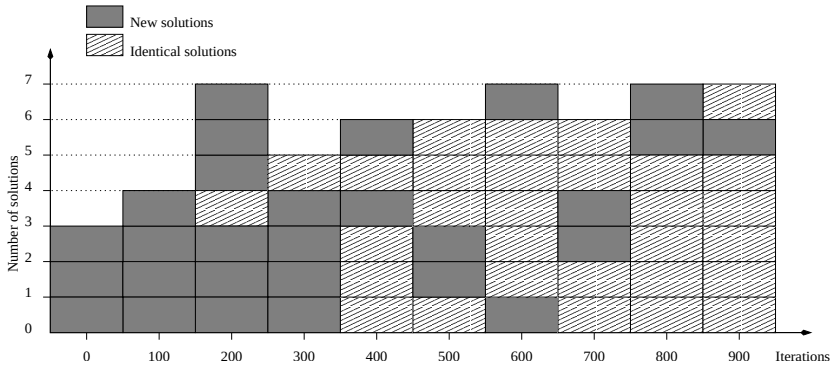

Fig. 3. Evolution of the number of efficient solutions for *gdb11* without local search procedure

to the efficient front. For example, the fifth solution at the 200^{th} iteration is in all efficient fronts until the end. The first solution at iteration 300 stays in the population for 200 iterations more and is replaced by a better solution at iteration 600. This recent solution remains in the population until iteration 900.

Figure 4 is identical to Figure 3 but the local search procedure is systematically applied to offspring (rate 100%). 700 iterations are performed before stopping the execution. More solutions are generated at iteration 100, 6 instead of 4 without local search.

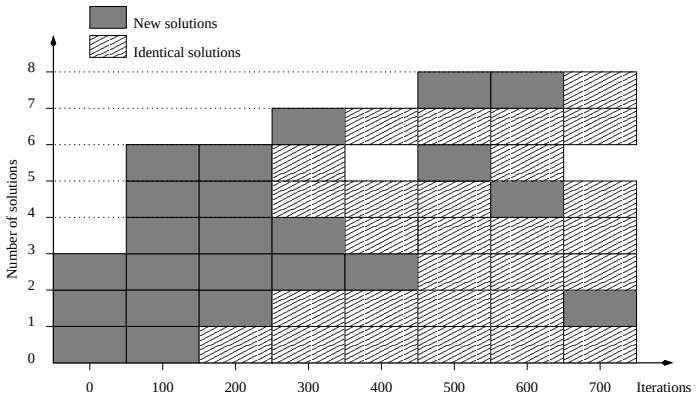

Fig. 4. Evolution of the number of efficient solutions for *gdb11* with local search

Less iterations are necessary and moreover less solutions appear during the execution. From iteration 400, 11 solutions appear in Figure 3 whereas only 6 solutions appear in Figure 4. Again, this shows the benefit of incorporating a

local search procedure into a MOGA. The convergence is accelerated by the local search.

5 Conclusion

Solving efficiently bi-objective real life problems is a challenging task. In addition to the total cost criterion, which is today well optimized by the best algorithms for the academic CARP, it is now possible to minimize the makespan simultaneously and this paper provides an efficient MOGA for this purpose.

In this approach, the implementation of the NSGA-II algorithm alone could not efficiently solve the bi-objective problem. Two key features are necessary: using good heuristics (Path-Scanning, Augment-Merge and Ulusoy's heuristics) in the initial population and adding a local search able to improve the solutions for both criteria. An intensive benchmark testing, with a strict comparison based on the best-known lower bounds and on a distance measure, proves the validity and the efficiency of the proposed approach.

It is foreseen to apply the MOGA to the town of Troyes. Although the algorithm is ready, this work cannot be done now, because the municipality must complete first its computerized representation of the street network: for instance, many forbidden turns are not yet entered.

References

1. A. Amberg and S. Voß. A hierarchical relaxations lower bound for the capacitated arc routing problem. In R.H. Sprague (Hrsg.), editor, *Proceedings of the* 35^{th} *Annual Hawaï International Conference on Systems Sciences*, pages DTIST02:1–10, Piscataway, 2002. IEEE.
2. J.M. Belenguer and E. Benavent. A cutting plane algorithm for the capacitated arc routing problem. *Computers and Operations Research*, 2003. To appear.
3. J.M. Belenguer, E. Benavent, and F. Cognata. Un metaheuristico para el problema de rutas por arcos con capacidades. In *Proocedings of the* 23^{th} *national SEIO meeting*, Valencia, Spain, 1997.
4. C.A. Coello Coello. An updated survey of ga-based multiobjective optimization techniques. *ACM Computing Surveys*, 32(2):109–143, 2000.
5. A. Corberan, E. Fernandez, M. Laguna, and R. Martí. Heuristic solutions to the problem of routing school buses with multiple objectives. *Journal of the Operational Research Society*, 53(4):427–435, 2002.
6. K. Deb. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation*, 7(3):205–230, 1999.
7. K. Deb. *Multi objective optimization using evolutionary algorithms*. Wiley, 2001.
8. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
9. M. Ehrgott and X. Gandibleux. *Multiobjective Combinatorial Optimization*, volume 52 of *International Series in Operations Research and Management Science*, pages 369–444. Kluwer, 2002.

10. B.L. Golden, J.S. DeArmon, and E.K. Baker. Computational experiments with algorithms for a class of routing problems. *Computers and Operations Research*, 10(1):47–59, 1983.
11. B.L. Golden and R.T. Wong. Capacitated arc routing problems. *Networks*, 11:305–315, 1981.
12. A. Hertz, G. Laporte, and M. Mittaz. A tabu search heuristic for the capacitated arc routing problem. *Operations Research*, 48(1):129–135, 2000.
13. S.C. Hong and Y.B. Park. A heuristic for bi-objective vehicle routing problem with time windows constraints. *International Journal of Production Economics*, 62:249–258, 1999.
14. P. Lacomme, C. Prins, and W. Ramdane-Chérif. Competitive memetic algorithms for arc routing problems. *Annals of Operations Research*, 2002. To appear. See also Research report LOSI-2001-01, Université de Technologie de Troyes, France.
15. P. Moscato. *New ideas in optimization*, chapter Memetic algorithms: a short introduction, pages 219–234. MacGraw-Hill, 1999.
16. Y.B. Park and C.P. Koelling. An interactive computerized algorithm for multicriteria vehicle routing problems. *Computers and Industrial Engineering*, 16:477–490, 1989.
17. A. Riise. Comparing genetic algorithms and tabu search for multi-objective optimization. In *Abstract conference proceedings*, page 29, Edinburgh, UK, July 2002. IFORS.
18. W. Sessomboon, K. Watanabe, T. Irohara, and K. Yoshimoto. A study on multi-objective vehicle routing problem considering customer satisfaction with due-time (the creation of pareto optimal solutions by hybrid genetic algorithm). *Transaction of the Japan Society of Mechanical Engineers*, 1998.
19. G. Ulusoy. The fleet size and mixed problem for capacitated arc routing. *European Journal of Operational Research*, 22:329–337, 1985.
20. E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.