ELSEVIER European Journal of Operational Research 89 (1996) 108–126

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

A heuristic algorithm for the Asymmetric Capacitated Vehicle Routing Problem

Daniele Vigo

Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna, Viale Risorgimento 2, 41036 Bologna, Italy

Received August 1993; revised July 1994

Abstract

We consider the *Asymmetric Capacitated Vehicle Routing Problem* (ACVRP), a particular case of the standard asymmetric *Vehicle Routing Problem* arising when only the vehicle capacity constraints are imposed. ACVRP is known to be NP-hard and finds practical applications, e.g., in distribution and scheduling. In this paper we describe the extension to ACVRP of the two well-known Clarke–Wright and Fisher–Jaikumar heuristic algorithms. We also propose a new heuristic algorithm for ACVRP that, starting with an initial infeasible solution, determines the final set of vehicle routes through an insertion procedure as well as intra-route and inter-route arc exchanges. The initial infeasible solution is obtained by using the additive bounding procedures for ACVRP described by Fischetti, Toth and Vigo in 1992. Extensive computational results on several classes of randomly generated test problems involving up to 300 customers and on some real instances of distribution problems in urban areas, are presented. The results obtained show that the proposed approach favourably compares with previous algorithms from the literature.

Keywords: Vehicle routing problem; Heuristic algorithms; Local search

1. Introduction

The *Vehicle Routing Problem* (VRP) can be defined as that of determining the set of routes for a vehicle fleet of known dimension, so that the whole demand associated to a customers' set is satisfied, and a given cost function minimized. Many additional constraints and assumptions can be added to this basic framework in order to take into account some aspects of real-life distribution and scheduling problems. For complete surveys on VRP see, e.g., Bodin, Golden, Assad and Ball [1], and Laporte [13].

In this paper we consider the *Asymmetric Capacitated Vehicle Routing Problem* (ACVRP), being a particular case of VRP In which all the vehicles are identical and the only imposed constraints require that, for each vehicle route, the total customers' demand does not exceed the given vehicle capacity. Moreover, as happens in some applications (e.g., the distribution in urban areas, where distances depend on one-way directions imposed on roads, as well as some scheduling problems with sequence dependent setup times and resource constraints), the distance matrix associated to customers' set is asymmetric.

ACVRP can be formulated as a graph theory problem on a complete directed graph $G=(V,A)$. Vertex 1 corresponds to the depot, where k identical vehicles with capacity $D_{\max}$ are stationed. Each vehicle can carry out at most one route. A non-negative demand d_j is associated with each vertex $j = 2, \ldots, n$, corresponding to the customers, while the depot has a fictitious demand $d_1 = 0$. With each arc $(i,j) \in A$, $i \neq j$, a non-negative cost, c_{ij} ($c_{ii} = 0$ for each $i = 1, \ldots, n$) is associated. ACVRP then calls for the determination of a collection of simple *circuits* (each corresponding to a vehicle route) such that: (i) each circuit passes through the depot; (ii) each customer is visited once by exactly one circuit; (iii) the total load of each circuit does not exceed vehicle capacity, $D_{\max}$. The objective is to minimize the number of circuits used and then the total routing cost, defined as the sum of the cost of the arcs belonging to the circuits. Because of the problem objective, and in order to ensure feasibility, we assume without loss of generality that k (the number of available vehicles) is equal to the minimum number of vehicles sufficient to serve all the customers. In this case the first objective can be re-stated as the constraint imposing that exactly k circuits are used. The value k can be easily obtained by solving the Bin Packing Problem (BPP) associated to ACVRP, in which the bin capacity is equal to $D_{\max}$ and each item j has weight equal to the demand d_j of the corresponding customer. In spite of the fact that BPP is NP-hard in the strong sense, instances with hundreds of items can be optimally solved very effectively (see, e.g. Martello and Toth [16]).

ACVRP is known to be NP-hard in the strong sense. Indeed, it generalizes two other well-known NP-hard problems. The first is the Travelling Salesman Problem (TSP), arising when $D_{\max} = +\infty$ (and hence $k = 1$), while the second is the already mentioned BPP. Exact algorithms for the capacitated VRP with symmetric distance matrix (CVRP) can be found in Christofides, Mingozzi and Toth [5], Fisher [11], and Cornuejols and Harche [6], while branch-and-bound exact algorithms for the asymmetric case have been developed by Laporte, Mercure and Nobert [12], and Fischetti, Toth and Vigo [9]. Many effective heuristic algorithms for the capacitated VRP have also been proposed so far. These methods have been developed for the symmetric case, but some algorithms can be extended to solve asymmetric problems. For a survey on heuristic algorithms for CVRP see Christofides, Mingozzi and Toth [4], Bodin, Golden, Assad and Ball [1], and Laporte [13].

In this paper we present a new heuristic algorithm specially developed for ACVRP. The development of this heuristic has been primarily motivated by the poor quality of the solutions that can, in general, be obtained by applying traditional methods to asymmetric instances. The algorithm starts with an initial infeasible solution and determines the final set of vehicle routes through an insertion procedure, intra-route and inter-route arc exchanges. The initial infeasible solution is obtained by using the additive bounding procedures for ACVRP described in Fischetti, Toth and Vigo [9]. The present paper is organized as follows. In Section 2 the extension to ACVRP of the two well-known Clarke–Wright [3] and Fisher–Jaikumar [10] heuristic algorithms is briefly discussed. The new heuristic algorithm, called AV in the sequel, is described in Section 3, while the results of computational testing on some classes of randomly generated problems and on some real-world distribution problems are presented in Section 4.

2. Extension of existing heuristic algorithms

A great number of heuristic solution strategies have been proposed for CVRP. Though these algorithms have been developed for the symmetric (or even Euclidean) case, some can be adapted also for solution of the asymmetric case. In the following we will discuss the extension to ACVRP of two of the most important, and successful, heuristic algorithms: the Clarke–Wright [3] savings algorithm and the Fisher–Jaikumar [10] optimization based method.

The Clarke–Wright algorithm is one of the first attempts made towards solution of CVRP. Due to its easiness of implementation and speed it has been widely used as a basis in many commercial routing packages. The algorithm starts with a solution in which each customer is visited by a separate route. Note that this solution is infeasible since it requires more than k vehicles. Routes are then iteratively combined by considering the ‘saving’, in terms of routing cost, that can be achieved by serving two customers on the same route instead of leaving them on separate ones. The saving obtained by visiting customers i and j in sequence in the same route is:

$$s_{ij} = c_{1i} + c_{i1} + c_{1j} + c_{j1} - (c_{1i} + c_{ij} + c_{j1}) \\ = c_{i1} + c_{1j} - c_{ij}. \qquad (1)$$

The extension of the Clarke–Wright algorithm to asymmetric instances poses no difficulty. Indeed, also in this case the saving can be computed using Eq. (1). The only difference from the symmetric case is that the routes are now oriented. Hence a link (i,j) can be considered as a candidate for the merging of two different partial routes only if vertex i (resp. j) is the last of a route and vertex j (resp. i) is the first of the other, thus halving the merging possibilities with respect to CVRP.

It should also be noted that the Clarke–Wright method does not allow for the control of the number of routes of the final solution. The solution found for a given instance can, in fact, require more than k routes to serve all the customers, hence being infeasible. From a practical point of view, the routing cost of the solution obtained with the Clark–Wright algorithm, as well as the probability that this solution is feasible, is strongly related to the number of route mergings executed. It is then evident that the route orientation arising in the asymmetric case, and the consequent reduction of possible route merging combinations, can greatly reduce the effectiveness of this method in facing asymmetric problems, both in terms of overall routing cost and of the number of feasible solutions found.

An effective way to improve the performance of the Clarke–Wright algorithm is to extend the parametric saving function to ACVRP as proposed by Paessens [17] for the symmetric case. The parametric saving associated with customers' sequence (i,j) is defined as:

$$s'_{ij} = c_{i1} + c_{1j} - g \cdot c_{ij} + f \cdot |c_{1i} - c_{j1}| \qquad (2)$$

where $g \in]0,3]$ and $f \in [0,1]$ are the two parameters. The Clarke–Wright algorithm with modified saving can be run with different choices of the parameters g and f, thus obtaining different solutions. Our computational experience has shown that this parametric technique considerably improves the performance of the Clarke–Wright algorithm, mainly with respect to the number of feasible solutions found. In particular, by using the M4 parameter combination proposed by Paessens [17], which leads to eight different runs for each instance, we observed a considerable improvement in the number of feasible solutions found compared to the non-parametric version of the algorithm, with a growth of one order of magnitude in the overall computing time.

Fisher and Jaikumar proposed in [10], a cluster-first-route-second algorithm for CVRP. The algorithm is based on the reformulation of the problem as a non-linear Generalized Assignment Problem (GAP) that generates a feasible allocation of customers to routes, and whose objective function takes into account the cost of sequencing the customers in each route. In the proposed approach, the non-linear objective function is approximated by a linear one, and the solution method is then decomposed into two phases. During the first phase the customers are partitioned into subsets by solving a GAP with a linearized objective function. The obtained subsets are feasible with respect to the capacity constraint. The second phase-builds the final solution by sequencing customers into each route through a TSP algorithm.

One possible way to construct a linear approximation of the objective function is described in [10] and is based on the determination of k *seed* customers (or points), one for each vehicle. The coefficients of the linearized objective function are then defined as the cost of inserting each customer into the simple route from the depot to a seed, and back to the depot. Because of the strong influence of the seeds on the assignment of customers to routes, and therefore on the final solution obtained with the overall algorithm, their choice is a very critical step for the Fisher–Jaikumar heuristic.

The seed selection rule proposed in [10] has been developed for the solution of instances with euclidean distances, hence its straightforward extension to asymmetric problems can lead to un-

Table 1

```
procedure STEP – two($S_3$,$S_2$,$k$,$c$,$\beta_2$):
begin $S_2 := \emptyset$;
  while $|S_2| < 2k$ do
    begin
      $j^* := \arg\max_{j \in S_3} \left\{ \left( \frac{c_{1j} + c_{j1}}{2} \right) + \beta_2 \sum_{i \in S_2} \left( \frac{c_{ij} + c_{ji}}{2} \right) \right\}$;
      $S_3 := S_3 \setminus \{j^*\}$; $S_2 := S_2 \cup \{j^*\}$
    end
  end.
```

predictable results. In the following we describe an adaptation of the seed selection rule proposed by Savelsbergh [18] for CVRP, that can be more successfully used in the solution of ACVRP. This method is based on the construction of a first set of candidate seeds with cardinality $3k$ that, in two successive steps, is gradually reduced to the final route seed set, with cardinality k. The first set S_3 contains the first $3k$ customers chosen in decreasing order of a 'difficulty degree', defined as the weighed combinations of the customer average distance from the depot and its demand: $\delta_i := \frac{1}{2}(c_{1i} + c_{i1}) + \beta_3 d_i$, where β_3 is the weighing parameter.

The first refinement of the candidate seed set is performed by extracting from it the $2k$ customers that are farthest from each other. To this end the procedure presented in Table 1 is used, where β_2 is an appropriate weight.

At each iteration of this procedure, the seed that has the maximum value of the weighed sum of the average distance from the depot and from the other seeds already in S_2, is selected from S_3.

For each customer in S_2 the 'associated load' is then defined, i.e. the total demand that would be assigned to that seed in the solution of a GAP with infinite capacity. The final set S_1 is determined by selecting the k customer having the larger associated load. This quantity can be determined with the procedure shown in Table 2, where each customer is associated with the seed for which the insertion cost is a minimum.

Weighing parameters β_1, β_2 and β_3 can be used to tune the seed selection method for different problem instances. As an example, when the capacity constraint is particularly tight, a high β_3 value enhances customer demands in the choice of the initial seed. In our extensive computational testing we have not been able to define general criteria for the choice of weighing parameters. Nevertheless, good results for non-triangularized instances have been obtained using, $\beta_1 = 5$, $\beta_2 = 5$ and $\beta_3 = \frac{5}{3}$, while for triangularized instances good values were $\beta_1 = 1$, $\beta_2 = 2$ and $\beta_3 = \frac{1}{2}$.

3. A new heuristic algorithm for ACVRP

In this section we propose a new heuristic algorithm, called AV, explicitly tailored for ACVRP. The development of this algorithm has been primarily motivated by the poor quality of the solutions obtainable, in many cases, by using the traditional methods described in Section 2 for the solution of asymmetric problems. The structure of the proposed method is as follows: AV starts from an initial, possibly infeasible, solution

Table 2

```
procedure STEP – THREE($S_2$,$S_1$,$k$,$c$,$\beta_1$):
begin
  for each customer $i$ do $L(i) := 0$;
  for each customer $i$ do
    begin
      $s^* := \arg\min_{s \in S_2} \{\min(c_{1i} + \beta_1 c_{is} + c_{s1} - c_{1s}, c_{i1} + \beta_1 c_{si} + c_{1s} - c_{s1})\}$;
      $L(s^*) := L(s^*) + d_i$
    end;
  $S_3 := \emptyset$;
  sort $L$ in decreasing order, and insert in $S_3$ the elements corresponding to the first $k$ values of $L$
end.
```

and tries to obtain a final feasible set of routes through customer movements and arc exchanges. In the following we describe the AV heuristic, separately focusing on its three main steps: the inital solution construction, the insertion and the refining step.

3.1. Initial solution construction

Any collection of joint simple circuits, visiting each customer once, and the depot k times (see Fig. 1 for an example), can be used as a starting solution for AV heuristic.

Given such an initial solution let $\pi(i)$ (resp. $\sigma(i)$) be the customer preceding (resp. following) customer i in the solution, $\rho(i)$ the circuit to which customer i belongs, and $D(h)$ the total load associated to circuit h. Keeping in mind the definition of ACVRP, a circuit h is infeasible either if its total associated load, $D(h)$, exceeds the vehicle capacity, $D_{\max}$, or if it is disconnected from the depot.

A possible way to obtain starting solution is to solve a relaxation of the problem. This choice is motivated by the fact that the solution of a good relaxation often has a high degree of information

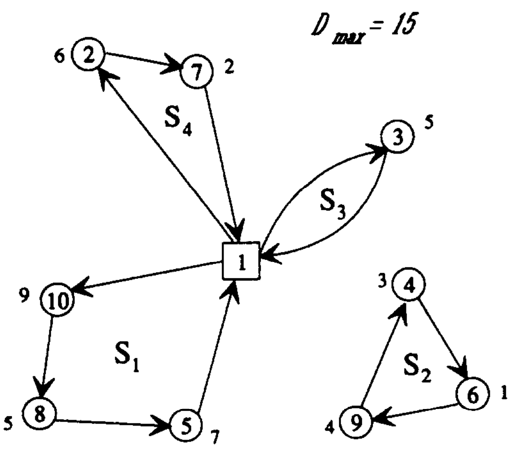

Fig. 1. An example of an admissible initial solution for an instance with $n = 10$, $k = 3$ and $D_{\max} = 15$. The demand of each customer is given besides the relative representative point. This initial solution contains two infeasible circuits: circuit 1 is overloaded, while circuit 2 is disconnected from the depot.

on the optimal solution structure, hence it can lead to better results with the AV algorithm. Note also that by using a relaxation the initial solution is in general infeasible.

In the AV algorithm the initial solution is obtained by using the additive lower bounding procedures ADD_DISJ and ADD_FLOW described in Fischetti, Toth and Vigo [9] that, according to our computational experience, provide very good starting solutions. These procedures solve, in the additive fashion proposed by Fischetti and Toth [7], different relaxations of ACVRP, each exploiting a particular substructure of the problem, and produce a sequence of non-decreasing lower bounds on the value of the optimal solution. The computational complexity of the overall additive procedure is $O(n^3)$. As described in Fischetti, Toth and Vigo [9], the solution obtained at the end of each iteration of the additive lower bounding procedures is a collection of $h \geqslant k$ circuits disjoint with respect to the customers and passing k times through the depot. Each of these solutions can directly be used as a starting solution for AV. Moreover, since the obtained lower bound value has proved to be very tight (see Fischetti, Toth and Vigo [9]), its value can be used to compute a good estimate of the quality of the solution constructed by AV, when no optimal solution is available.

3.2. Insertion procedure

The insertion procedure is executed when the initial solution is infeasible and tries to build a feasible one. In this phase customers are iteratively removed from infeasible circuits and inserted in the best position of a feasible circuit (route). Possible insertions are considered only if they do not overload the target route.

For each vertex i, belonging to an infeasible circuit, the overall 'cost', θ_i^j, of its insertion after vertex j, belonging to a feasible one, is evaluated according to the following (see Fig. 2 for an illustration):

$$\theta_i^j := \left(c_{ji} + c_{i\sigma(j)} + c_{\pi(i)\sigma(i)} - c_{\pi(i)i} - c_{i\sigma(i)-c_{j\sigma(j)}}\right) - \alpha_1 \Delta_i, \tag{3}$$

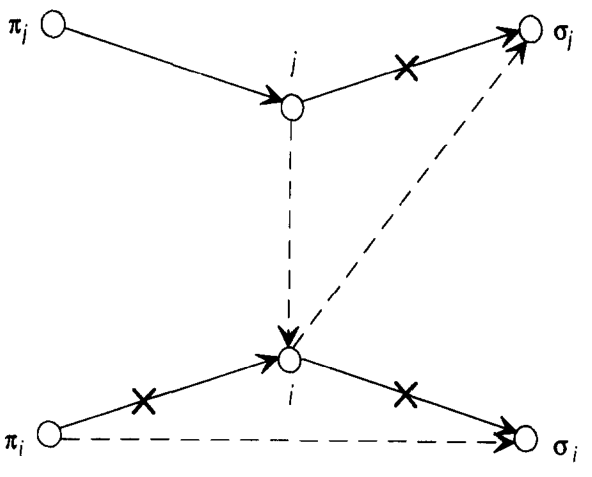

Fig. 2. Insertion of customer i after customer j. Removed arcs are indicated with a cross, while new arcs are drawn with hatched lines.

where Δ_i, is the reduction of the overload of circuit $\rho(i)$ due to the removal of customer i, and α_1 is an appropriate weighing parameter. If the infeasible circuit is disconnected from the depot we have $\Delta_i = d_i$, otherwise $\Delta_i = \min\{D(\rho(i)) - D_{\max}, d_i\}$. At each iteration of the insertion phase, the best insertion (corresponding to the minimum value of θ_i^j) is executed. The procedure is iterated until either no feasible insertion exists, or the circuit collection is feasible. This procedure can easily be extended to consider not only single vertices but sequences of consecutive vertices to be moved from an infeasible route to a feasible one. In Table 3 we give a Pascal-like outline of the insertion phase.

The computational complexity of each iteration of INSERT procedure is $O(n^2)$, since at each iteration $O(n^2)$ possible insertions are considered and the insertion cost and feasibility check of the new routes can be done in constant time. According to our computational experience, by using high values for parameter α_1 it is possible to obtain, in general, a higher quantity of feasible solutions at the end of the insertion step. However, in some cases the cost of the solutions obtained with higher values of the parameter become worse. This can be explained by the fact that with high α_1 values the component of θ_i^j due to the infeasibility decrease prevails with respect to the routing cost in the evaluation of possible insertions.

Table 3

```
procedure INSERT($\sigma,\pi,\rho,D,D_{\max},\alpha_1$):
begin
 repeat
   $\theta_{\min} := +\infty$;
   $M := \max\{D_{\max} - D(h)$, for each circuit $h$ connected to the depot$\}$;
   comment: $M$ is the maximum load that can be feasibly inserted into a circuit;
   for each sequence of consecutive vertices $S := \{i_1,\ldots,i_q\}$ of an infeasible circuit $h$ such that $L := \sum_{i \in S} d_i < M$ do
     for each vertex $j$ such that $D(\rho(j)) + L \leqslant D_{\max}$ do
       begin
         evaluate the cost $\theta_S^j$ of the insertion of $S$ after vertex $j$;
         if $\theta_S^j < \theta_{\min}$ then
           begin
             $\theta_{\min} := \theta_S^j$; $j^* := j$; $S^* := S$; $h^* := h$; $L^* := L$
           end
       end;
   if $\theta_{\min} < +\infty$ then
     begin
       insert sequence $S^*$ after vertex $j^*$ and remove it from $h^*$;
       $D(h^*) := D(h^*) - L^*$;
       $D(\rho(j^*)) := D(\rho(j^*)) + L^*$
     end
 until (all circuits are feasible) or ($\theta_{\min} = +\infty$)
end.
```

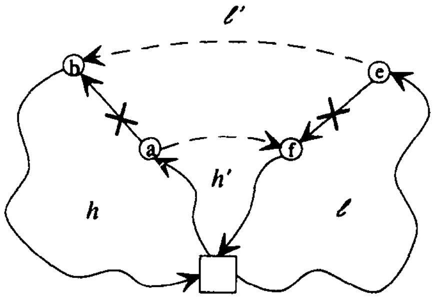

Fig. 3. The only possible two-arc exchange for the asymmetric case. Removed arcs are indicated by a cross, while those added are drawn with hatched lines.

If algorithm AV does not return a feasible solution, procedure INSERT can be restarted, choosing either a different initial solution or a new (higher) value for parameter α_1. For example, as new initial solution, one of those obtained in the intermediate iterations of the additive procedure can be used. Nevertheless, this restart is not in general necessary since the possibly residual infeasibility can be removed also by the refining step. In all the instances examined in Section 4, algorithm AV has been able to determine a feasible solution in a single iteration.

3.3. Refining procedure

A refining procedure is applied to the solution obtained through the insertion step. This procedure is composed of a succession of inter-route and intra-route arc exchanges and customers' movements like those described in Lin [14], Lin and Kernighan [15], Christofides and Eilon [2], and Savelsbergh [18], that proved to be very effective on various optimization problems.

As a first step of the refining procedure the exchange of arc pairs belonging to different routes is considered. An example of these interroute exchanges is illustrated in Fig. 3. It is easy to verify that this is the only possible two-arc exchange type for ACVRP. The final 2-opt solution is obtained by iteratively evaluating the routing cost and the possible infeasibility reduction produced by each exchange of two arcs, and performing the best exchange among all those that produce a positive cost reduction (also called *active exchanges*). Arc exchanges that produce infeasible circuits starting from feasible ones are not considered. The procedure is iterated until no active exchanges are found.

Let η_h be the overload of route h, defined as $\eta_h := \max\{0, D(h) - D_{\max}\}$. The cost θ_{ab}^{ef} of the exchange of arc (a,b), and arc (e,f) belonging, say, to route h and l, respectively, with arcs (a,f) and (e,b), is computed as the weighed sum of the routing cost term and of the infeasibility reduction term, according to the following:

$$\theta_{ab}^{ef} := c_{af} + c_{eb} - c_{ab} - c_{ef} + \alpha_2(\eta_{h'} + \eta_{l'} - \eta_h - \eta_l) \quad (4)$$

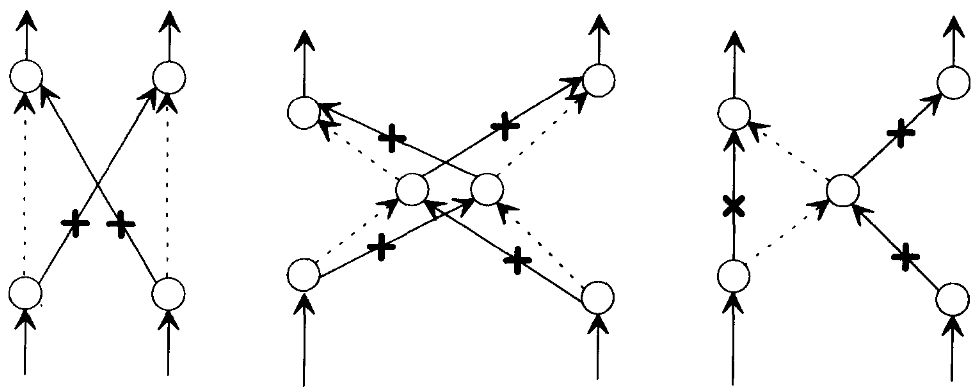

Fig. 4. Examples of the Relocate, Exchange and Cross exchanges. Removed arcs are indicated by a cross, while those added are drawn with hatched lines.

where h' and l' are the two new routes produced by the exchange as illustrated in Fig. 3.

At each iteration $O(n^2)$ exchanges are considered. The feasibility check and the computation of η for the new routes can be executed in constant time through parametric labelling techniques. Hence the computational complexity of a single iteration is $O(n^2)$.

The second step of the refining procedure considers an extension of the exchanges, called *Relocate*, *Exchange* and *Cross*, described in Savelsbergh [18] and illustrated in Fig. 4, in which sequences of consecutive customers are iteratively moved from one route to another. We have slightly extended the neighbourhoods proposed by Savelsbergh in order also to consider arc pairs belonging to the same route as well as the infeasibility reduction term in the exchange cost, as for the previous step. Moreover, instead of applying the three exchange procedures in sequence we have used a single, larger neighbourhood called REC in which all the Relocate, Exchange and

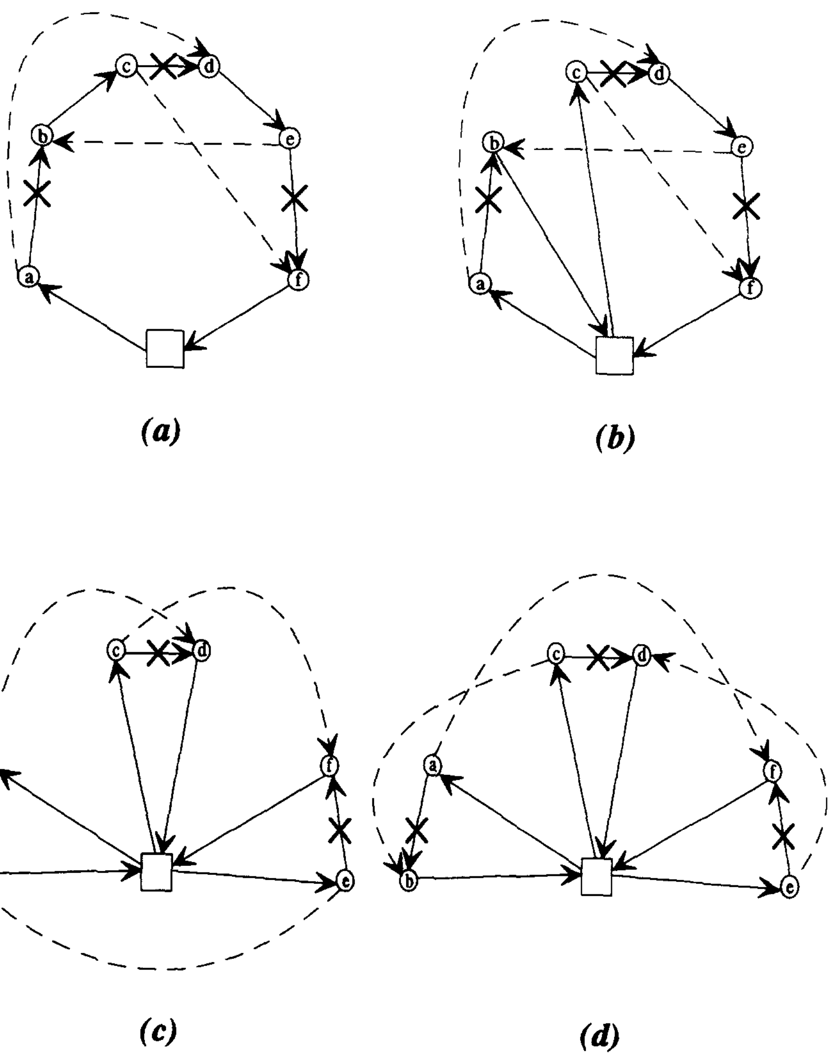

Fig. 5. The only possible three-arc exchanges for the asymmetric case: in (a) all three removed arcs belong to the same circuit and the exchange produces a single circuit, in (b) the removed arcs belong to two circuits and the exchange produces two circuits, while in (c) and (d) the removed arcs belong to three different circuits and the exchange produces three circuits.

Table 4a
Problems of Class I with n between 10 and 90. Average results over 5 instances, $\alpha_1 = 10$, $\alpha_2 = 1000$, $\mu = \frac{1}{5}n$; times are given in Digital VAXstation 3100 CPU seconds

n	α	k	% ld	opt /lb	APCW		APCW + ref		AFJ		APJ + ref		AV		AV + ref	
					%sol	t	%sol	t	%sol	t	%sol	t	%sol	t	%sol	t
10	0.15	4	87	*	125.6 (3)	0.12	123.1 (3)	0.15	136.6	0.07	132.0	0.11	129.5 (4)	0.06	134.7	0.09
	0.20	3	82	*	120.8	0.12	115.8	0.16	137.2	0.06	125.8	0.09	115.1	0.04	114.1	0.06
	0.25	3	83		108.5 (4)	0.12	104.3 (4)	0.16	125.0	0.04	117.9	0.08	107.5 (4)	0.05	102.7	0.08
	0.50	2	83		117.4	0.13	108.8	0.17	132.3	0.03	120.2	0.06	129.9	0.03	106.6	0.05
	0.75	2	62		107.3	0.13	101.9	0.17	125.5	0.03	108.1	0.06	105.0	0.03	102.7	0.05
20	0.15	5	88	*	165.1 (4)	0.47	149.6 (4)	0.63	188.8	0.26	164.9	0.54	159.0	0.11	148.6	0.28
	0.20	4	88	*	192.0	0.48	176.9	0.67	203.6	0.21	177.1	0.41	166.9	0.11	143.9	0.29
	0.25	4	81		133.0 (4)	0.50	125.4 (4)	0.77	154.6	0.13	136.8	0.33	140.2	0.11	126.4	0.24
	0.50	2	91		185.3	0.50	153.5	0.82	138.3	0.10	133.7	0.23	117.8	0.08	115.0	0.18
	0.75	2	64		157.5	0.50	119.0	0.94	143.5	0.09	122.7	0.28	193.5	0.07	103.5	0.15
30	0.15	5	97	*	188.1 (2)	1.12	167.4 (2)	1.91	253.8	1.28	237.3	1.80	180.0 (3)	0.21	165.8	0.78
	0.20	4	91	*	199.6 (3)	1.09	185.3 (3)	1.87	232.4	0.65	195.0	1.32	156.4	0.19	150.6	0.61
	0.25	4	83		176.3	1.12	159.4	1.95	209.6	0.22	150.7	0.97	130.2	0.21	122.5	0.61
	0.50	2	94		213.2	1.13	163.8	2.64	160.0	0.18	148.6	0.63	132.3	0.17	119.5	0.55
	0.75	2	65		182.6	1.13	136.2	2.97	161.1	0.17	126.0	0.78	124.9	0.17	111.3	0.42
40	0.15	6	87	*	204.3	2.01	169.0	4.24	265.6	1.42	201.2	3.34	148.9	0.36	138.1	118
	0.20	5	84	*	188.7	1.99	164.0	3.58	284.3	0.58	207.2	2.49	148.9	0.32	136.9	1.17
	0.25	4	87		204.7	2.02	166.5	3.97	247.8	0.57	174.3	2.85	130.3	0.32	124.9	1.11
	0.50	2	95		255.5	2.04	205.1	4.02	169.9	0.45	163.9	1.14	114.5	0.29	113.7	0.86
	0.75	2	66		226.5	2.03	170.2	3.55	169.8	0.41	150.6	1.50	106.3	0.32	105.2	0.82
50	0.15	6	92	*	249.1	3.21	216.8	5.84	349.3	2.89	253.3	6.12	185.2	0.52	161.8	2.11
	0.20	5	87	*	262.8	3.19	198.5	8.22	347.1	1.49	227.8	6.26	141.3	0.41	137.5	1.69
	0.25	4	90		268.2	3.22	224.8	7.75	289.8	0.95	204.5	6.77	135.8	0.54	119.8	1.79
	0.50	2	96		310.8	3.24	226.5	9.15	180.2	0.60	175.4	1.67	122.7	0.39	117.8	1.57
	0.75	2	66		279.1	3.24	174.5	9.43	175.4	0.49	151.0	1.85	108.7	0.40	107.9	1.18
60	0.15	6	93	*	287.8 (2)	4.66	235.4 (2)	10.09	367.5	4.36	271.4	10.74	198.7	0.80	172.9	3.81
	0.20	5	88	*	292.9	4.68	226.1	12.68	344.1	2.72	250.1	9.30	169.9	0.75	148.2	3.31
	0.25	4	91		280.5	4.71	212.4	14.47	315.1	1.45	253.2	6.96	139.8	0.66	129.0	2.49
	0.50	2	97		323.8	4.73	242.7	14.25	179.7	0.91	178.0	2.77	119.6	0.75	119.3	2.18
	0.75	2	66		296.7	4.73	191.1	15.87	187.4	0.82	164.4	3.28	105.2	0.75	105.2	1.49
70	0.15	6	95	*	260.9 (1)	6.47	234.5 (1)	13.27	401.2	5.47	325.5	15.38	180.8	0.83	165.9	3.94
	0.20	5	90	*	299.9	6.51	239.2	17.86	363.3	3.34	265.8	13.80	142.1	1.05	137.0	3.61
	0.25	4	92		317.3 (3)	6.52	211.2 (3)	21.64	303.3	2.41	259.8	11.88	139.3	0.99	122.8	3.33
	0.50	2	97		337.3	6.55	239.0	24.50	188.3	1.20	186.8	3.13	128.0	1.01	125.6	3.28
	0.75	2	66		304.2	6.53	194.7	23.78	187.0	1.18	168.1	6.08	108.5	1.15	108.0	3.04
80	0.15	6	98	*	– (0)	–	– (0)	–	430.4	7.96	400.1	16.47	208.8	1.34	190.0	6.28
	0.20	5	91	*	411.9 (3)	8.56	277.3 (3)	36.48	395.7	2.45	302.8	17.43	168.4	1.35	147.1	5.87
	0.25	4	93		384.3	8.58	250.5	35.81	329.3	2.03	279.5	13.91	164.9	1.37	142.5	6.22
	0.50	2	98		456.8	8.67	279.7	41.52	199.9	1.26	197.9	4.19	122.7	1.05	117.1	4.36
	0.75	2	66		350.2	8.66	238.4	27.78	200.1	1.36	184.1	5.38	116.5	1.03	110.4	4.05
90	0.15	6	96	*	357.7 (1)	10.73	289.9 (1)	27.73	500.6	9.13	396.3	30.96	220.2 (4)	1.64	209.1	9.44
	0.20	5	92	*	463.0 (4)	10.98	338.7 (4)	44.94	433.1	6.30	302.4	33.39	163.8	1.33	149.5	6.20
	0.25	4	94		388.0	10.93	294.4	39.03	366.9	5.03	310.5	23.04	173.6	1.60	143.7	8.45
	0.50	2	98		529.8	11.06	334.1	57.07	195.9	1.46	195.9	4.84	123.2	1.39	122.8	4.96
	0.75	2	66		397.1	11.10	225.8	44.71	196.5	1.61	181.4	9.06	108.5	1.63	108.5	4.78

Table 4b
Problems of Class I with n between 100 and 300. Average results over 5 instances, $\alpha_1 = 10$, $\alpha_2 = 1000$, $\mu = \frac{1}{5}n$, times are given in Digital VAXstation 3100 CPU seconds

n	α	k	% ld	opt /lb	APCW		APCW + ref		AFJ		AFJ + ref		AV		AV + ref	
					%sol	t	%sol	t	%sol	t	%sol	t	%sol	t	%sol	t
100	0.15	7	92	*	373.6 (3)	13.63	280.2 (3)	58.22	560.1	8.35	407.9	37.96	196.6 (4)	2.04	210.7	12.92
	0.20	5	93	*	447.0	13.70	308.5	72.43	486.5	7.15	355.3	40.21	157.2	1.91	145.5	9.83
	0.25	4	95		531.8 (3)	13.86	353.6 (3)	64.88	405.3	3.19	355.0	25.52	126.9	1.78	123.7	6.60
	0.50	2	98		500.1	13.86	352.4	64.89	216.5	2.56	215.8	7.45	119.1	1.57	119.1	5.63
	0.75	2	66		439.6	13.78	263.6	57.39	222.9	1.95	201.5	12.84	117.5	1.43	114.7	6.29
150	0.15	7	89	*	598.5	32.13	422.0	157.70	701.6	17.53	429.9	152.04	232.3	4.66	178.2	39.86
	0.20	5	95	*	744.4 (3)	32.07	466.7 (3)	241.65	595.0	11.24	513.7	75.49	169.7	4.27	158.5	19.37
	0.25	4	96		743.0 (1)	32.26	463.2 (1)	168.29	478.0	11.20	417.6	73.95	148.9	4.75	142.6	20.32
	0.50	2	99		793.5 (4)	32.26	444.3 (4)	291.36	250.3	4.26	230.3	15.67	125.2	4.86	125.2	15.55
	0.75	2	66		661.9	32.21	355.1	206.81	225.8	4.66	217.1	20.75	104.8	5.09	104.3	14.43
200	0.15	7	90	*	733.6	58.45	531.5	339.41	879.9	41.06	569.0	330.18	216.9	9.81	180.4	71.81
	0.20	5	96	*	1065.5 (2)	58.39	627.5 (2)	177.88	748.4	33.05	638.0	182.49	223.5	10.84	176.5	61.37
	0.25	4	97		1020.9 (1)	58.60	581.4 (1)	279.30	580.8	16.61	552.8	104.62	180.5	8.67	157.5	55.66
	0.50	2	99		1194.6 (4)	59.30	656.7 (4)	275.74	275.3	7.98	274.8	35.55	141.7	10.25	138.0	46.39
	0.75	2	66		870.7	59.10	465.9	416.82	275.2	7.64	258.1	41.33	124.4	11.89	117.7	35.77
250	0.15	7	91	*	1484.0 (4)	93.49	808.2 (4)	174.28	1329.5	56.31	904.7	612.87	261.1	19.13	225.9	108.76
	0.20	5	97	*	1589.1 (1)	93.28	1010.9 (1)	458.23	1075.4	44.56	966.7	307.36	261.4	17.11	207.9	115.02
	0.25	4	98		– (0)	–	– (0)	–	828.1	20.32	801.8	127.69	233.3	14.34	187.6	107.39
	0.50	2	99		1742.6 (1)	94.68	955.3 (1)	251.95	367.8	13.15	367.5	57.00	149.6	26.35	141.6	76.37
	0.75	2	66		1479.7	94.44	681.8	459.65	371.9	17.44	352.9	80.08	115.3	33.13	114.6	70.80
300	0.15	7	92	*	1764.9 (4)	136.23	1145.9 (4)	90.57	1877.6	57.21	1309.2	1015.86	267.8	23.00	205.0	140.76
	0.20	5	97	*	– (0)	–	– (0)	–	1532.9	59.00	1346.1	625.82	279.7	29.88	230.1	205.23
	0.25	4	98		1771.1 (1)	137.62	1302.2 (1)	151.49	1033.4	42.46	1004.0	222.91	204.8	26.36	174.2	157.41
	0.50	2	99		2152.7 (2)	138.08	1220.3 (2)	216.31	499.5	17.41	498.4	86.38	121.8	30.57	121.8	90.75
	0.75	2	67		2295.9	137.79	1125.1	207.07	515.1	21.16	494.0	141.40	123.2	32.41	120.2	92.09

Cross moves are considered in the same iteration. According to our computational experience the REC neighbourhood proved able to produce better solutions, with no significant increase in the computational effort, than those obtained by the sequence of three simple ones.

The final step of the refining procedure considers the intra-route and inter-route three-arc exchanges illustrated in Fig. 5. Since the computational complexity of each iteration is $O(n^3)$, it is necessary to reduce the running time of this step, for example by limiting the number of considered exchanges. This can be done by laying down that the maximum number of customers separating any pair of customers involved in an exchange is smaller than a prefixed value μ.

As to the choice of parameter α_2, used in the exchange evaluation, considerations similar to those made for α_1 in the previous section are valid. Nevertheless, observe that since procedure INSERT is almost always able to produce feasible solutions, in this case the choice of parameter α_2 is irrelevant.

4. Computational results

The asymmetric versions of the parametric Clarke–Wright and Fisher–Jaikumar algorithms, respectively called APCW and AFJ in the sequel, as well as the AV algorithm described in the previous section, have been implemented in FORTRAN, and run on a Digital VAXstation 3100 on several classes of test problems.

With the parametric Clarke–Wright algorithm we used the parameter choice M4 described by

Paessens [17] which, according to our computational experience produced good results without an excessive increase of the computing time. For each instance the algorithm is first run with the (g, f) pairs (1.0, 0.1), (1.0, 0.5), (1.4, 0.0), and (1.4, 0.5). Let (g', f') be the pair which produced the best solution value: four more runs of the algorithm are then executed with the (g, f) values $(g' - 0.1, f')$, $(g' + 0.1, f')$, $(g', f' - 0.1)$, and $(g', f' + 0.1)$.

In order to reduce the influence of the seed selection, algorithm AFJ has been applied 15 times to each instance, with different choices of parameters, β_1, β_2 and β_3. The results for this algorithm, reported in the following, are concerned with the best solution found within the 15 runs, while the computing time is the total one divided by 15. The GAP instances in AFJ have been solved using the FORTRAN code MTG, given in Martello and Toth [16]. Since the computing time needed for the exact solution of GAP was excessive, the maximum number of backtracking for MTG has been fixed at 100. According to our computational experience this value permits better results to be obtained with respect to the needed computational effort. For the solution of asymmetric TSP instances in AFJ the code described in Fischetti and Toth [8] has been used, with an imposed time limit of 100 seconds.

As to algorithm AV, the reported computing time includes computation of the additive lower bound. The described results have been obtained with the following values for the parameters: $\alpha_1 = 10$, $\alpha_2 = 1000$ and $\mu = \frac{1}{5}n$. This choice computationally proved to give good results compared with the required computing time.

We first compare the behaviour of the algorithms on pure asymmetric instances. To this end we examine a test problem class proposed by Laporte, Mercure and Nobert [12], where customer demands, d_j, and costs, c_{ij}, are uniformly random in the interval [0, 100] (rounded to the nearest integer). The vehicle capacity, $D_{\max}$, is defined by the following:

$$D_{\max} := (1 - \alpha) \max_{j \in V} \{d_j\} + \alpha \sum_{j \in V} d_j,$$

where α is a real parameter belonging to interval [0, 1]. The number of available vehicles is computed as

$$k := \left\lceil \sum_{j \in V} d_j / D_{\max} \right\rceil.$$

This produced a feasible problem in all the generated instances. As observed in Fischetti, Toth and Vigo [9] larger values of α produce larger $D_{\max}$, hence smaller k, while no monotone correlation between α and the *average percentage load* of a vehicle, defined as

$$100 \sum_{j \in V} d_j / (k D_{\max}),$$

can instead be inferred. As in Laporte, Mercure and Nobert [12] we considered $\alpha = 0.25$, 0.50 and 0.75, producing $k = 4$, 2, and 2, respectively. We have also considered $\alpha = 0.15$ and $\alpha = 0.20$, which produce instances with a higher number of vehicles.

Tables 4a and 4b compare the performance of algorithms APCW, AFJ and AV on problem instances with n between 10 and 300. For each value of n and α the tables give the average, computed over 5 instances, number of required vehicles (k), the average percentage load of each vehicle (%ld), and for each algorithm the following average results:

- %sol: the percentage ratio of the solution value obtained using the algorithm and the value of the optimal solution, computed through the exact algorithm described in Fischetti, Toth and Vigo [9]. For instances marked with an asterisk in the opt/lb column the solution value is compared with the additive lower bound value;
- t: the overall computing time, in DIGITAL VAXstation 3100 CPU seconds.

The total number of generated instances is 350. Statistics for each algorithm refer to the successfully solved instances only, whose number (when different from 5) is reported in parenthesis. For algorithms APCW and AFJ the solution found and the total computing time after application of the same refining procedure described in Section 3.3 is also reported. As to algorithm AV the average results obtained at the end of the

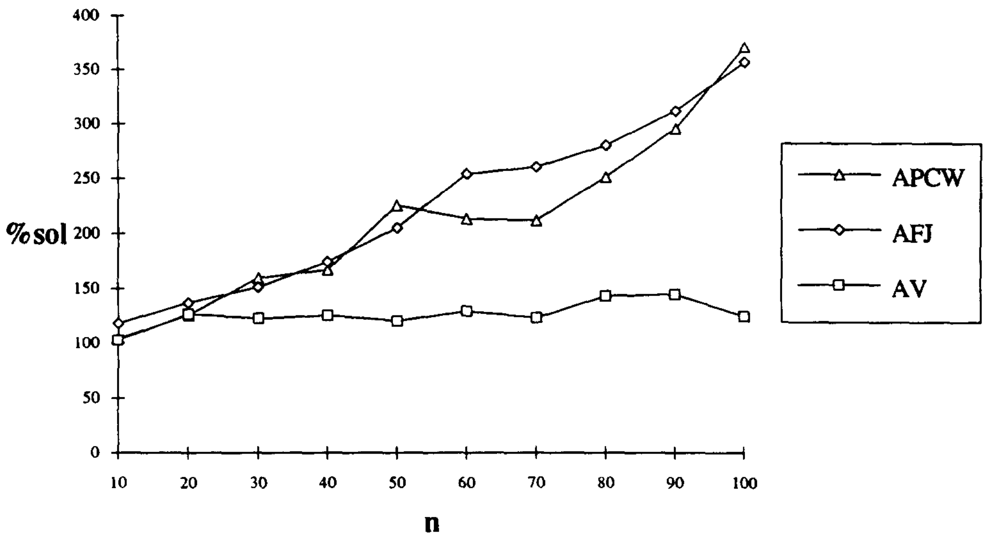

Fig. 6. The average percentage ratio of the solutions found by algorithms APCW, AFJ and AV with the refining step, on instances of Class I with $\alpha = 0.25$, and n between 10 and 100.

insertion phase, and at the end of the refining procedure are included.

Tables 4a and 4b clearly show that algorithm AV outperforms both APCW and AFJ. Indeed, for the instances of Class I, AV produced the best average solution in 67 out of 70 (n, α) pairs, both with and without the refining step. The greater effectiveness of algorithm AV can also be

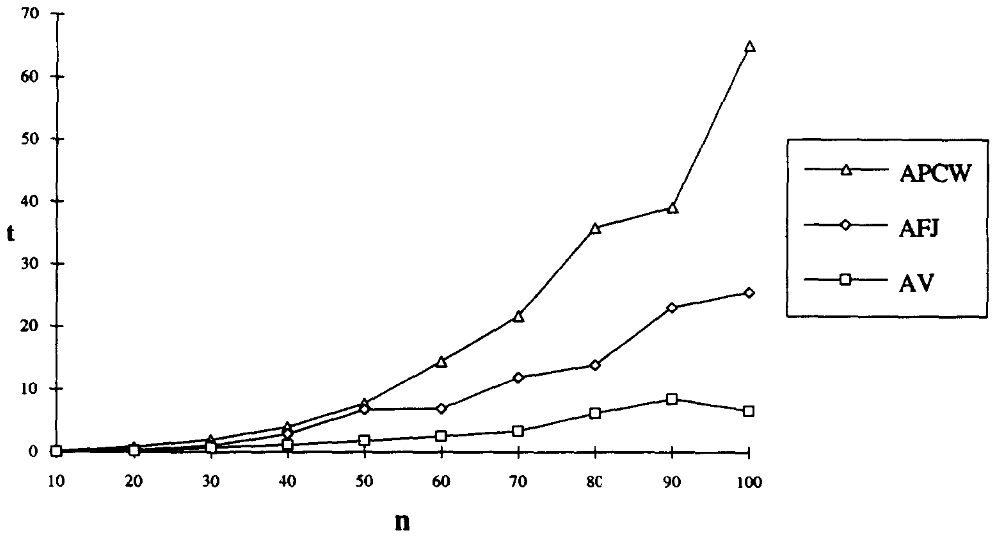

Fig. 7. The average computing time needed by algorithms APCW, AFJ and AV with the refining step, on instances of Class I with $\alpha = 0.25$, and n between 10 and 100.

Table 5a
Problems of Class II with n between 10 and 90. Average results over 5 instances, $\alpha_1 = 10$, $\alpha_2 = 1000$, $\mu = \frac{1}{5}n$, times are given in Digital VAXstation 3100 CPU seconds

n	α	k	% ld	opt /lb	APCW		APCW + ref		AFJ		AFJ + ref		AV		AV + ref	
					%sol	t	%sol	t	%sol	t	%sol	t	%sol	t	%sol	t
10	0.15	4	87	*	111.8 (4)	0.08	111.0 (4)	0.11	115.6	0.07	114.0	0.10	116.5 (4)	0.05	116.5	0.08
	0.20	3	82	*	114.2	0.07	112.5	0.11	119.2	0.04	113.1	0.07	110.8	0.04	110.8	0.06
	0.25	3	83		103.2	0.12	102.5	0.14	104.3	0.04	103.6	0.06	101.2	0.04	100.6	0.07
	0.50	2	83		101.7	0.12	101.7	0.14	105.3	0.03	103.2	0.06	101.9	0.03	101.9	0.04
	0.75	2	62		100.6	0.12	100.4	0.16	101.1	0.04	100.2	0.07	100.0	0.02	100.0	0.03
20	0.15	5	88	*	120.9 (4)	0.31	116.8 (4)	0.45	119.4	0.75	115.8	0.91	118.2 (4)	0.11	116.0	0.28
	0.20	4	88	*	119.0	0.31	115.3	0.45	120.5	0.53	117.7	0.68	113.8	0.10	113.8	0.21
	0.25	4	81		102.8 (4)	0.47	102.2 (4)	0.65	111.7	0.30	106.2	0.49	107.7	0.10	106.1	0.26
	0.50	2	91		110.1	0.48	107.0	0.64	108.8	0.08	106.9	0.23	102.2	0.07	101.5	0.20
	0.75	2	64		107.8	0.49	106.2	0.62	107.4	0.20	104.6	0.37	102.9	0.06	102.9	0.15
30	0.15	5	97	*	116.6 (1)	0.70	111.8 (1)	1.44	128.0 (4)	1.12	124.2 (4)	1.52	118.8 (4)	0.19	121.0	0.78
	0.20	4	91	*	113.1 (3)	0.70	110.4 (3)	1.25	127.3	1.01	119.5	1.67	113.1 (3)	0.18	112.3	0.68
	0.25	4	83		106.8	1.08	106.0	1.40	121.7	0.33	116.7	0.74	108.1	0.16	105.8	0.61
	0.50	2	94		109.9	1.10	106.3	1.56	111.7	0.14	110.8	0.54	102.5	0.14	102.3	0.41
	0.75	2	65		107.5	1.11	105.0	1.61	108.4	0.14	104.1	0.46	100.9	0.12	100.5	0.28
40	0.15	6	87	*	115.1	1.29	110.9	2.55	128.9	1.75	118.5	3.58	116.4	0.36	112.0	1.35
	0.20	5	84	*	112.7	1.29	109.9	2.34	128.7	1.25	115.3	3.13	112.1	0.33	107.9	1.28
	0.25	4	87		108.4	2.00	107.4	2.92	123.4	1.27	114.1	2.60	106.6	0.30	104.3	1.09
	0.50	2	95		118.0	2.01	114.7	3.19	112.2	0.27	110.0	1.03	104.8	0.25	104.5	0.82
	0.75	2	66		110.7	2.03	106.0	3.17	109.3	0.25	106.5	1.14	102.4	0.27	102.1	0.65

50	0.15	6	92	*	114.0 (4)	2.09	110.2 (4)	4.03	139.1	1.70	128.6	4.33	113.6	0.47	108.8	2.80
	0.20	5	87	*	115.5	2.09	110.7	4.43	134.3	0.84	120.2	3.88	107.1	0.38	104.4	2.02
	0.25	4	90		114.5	3.16	108.4	6.24	131.5	0.81	115.7	4.35	103.3	0.36	102.4	1.24
	0.50	2	96		116.3	3.21	111.6	6.22	113.7	0.26	112.3	1.98	102.0	0.37	101.9	0.81
	0.75	2	66		110.7	3.21	108.7	4.64	112.0	0.49	107.1	1.77	100.0	0.31	100.0	0.31
60	0.15	6	93	*	114.5 (1)	3.08	113.7 (1)	5.62	138.9	1.60	125.8	5.69	112.1	0.68	110.4	2.56
	0.20	5	88	*	114.6	3.05	109.6	7.02	137.2	0.69	121.8	6.50	106.7	0.63	105.3	2.25
	0.25	4	91		115.7 (4)	4.70	109.5 (4)	9.38	129.9	1.11	119.7	5.22	107.6	0.59	105.6	2.84
	0.50	2	97		118.1	4.70	112.9	8.41	116.7	0.39	111.5	3.44	101.6	0.49	101.5	1.41
	0.75	2	66		114.0	4.72	107.9	7.91	111.4	0.40	109.7	2.11	101.2	0.47	100.6	1.40
70	0.15	6	95	*	114.5 (1)	4.31	112.3 (1)	6.64	136.0	4.09	126.3	11.89	111.3	0.89	108.9	4.38
	0.20	5	90	*	121.1	4.36	112.8	11.14	131.3	2.10	117.4	11.59	106.2	0.80	105.2	3.66
	0.25	4	92		117.7	6.48	112.7	13.23	125.9	1.98	117.1	8.92	102.2	0.67	101.7	2.05
	0.50	2	97		120.7	6.47	115.3	12.78	116.4	0.44	114.2	4.56	103.6	0.69	103.2	2.72
	0.75	2	66		114.6	6.52	109.0	13.18	114.0	0.82	107.3	5.75	101.5	0.75	101.4	2.06
80	0.15	6	98	*	– (0)	–	– (0)	–	139.1	4.68	127.3	14.03	112.5	1.12	112.1	3.86
	0.20	5	91		116.1	5.82	112.2	12.35	133.2	3.61	120.1	14.92	105.5	1.08	103.6	4.60
	0.25	4	93		119.5	8.63	112.7	18.59	131.2	1.65	120.4	13.18	107.2	1.07	104.7	5.42
	0.50	2	98		120.5 (3)	8.66	113.6 (3)	19.10	116.3	0.53	114.6	5.29	102.7	1.16	102.0	4.06
	0.75	2	66		116.3	8.63	112.1	17.05	113.5	0.50	109.7	5.00	100.6	1.25	100.2	1.73
90	0.15	6	96	*	108.8 (1)	7.45	106.2 (1)	17.35	137.9	4.02	128.1	19.26	109.4 (4)	1.63	108.8	7.35
	0.20	5	92	*	118.2 (4)	7.46	112.7 (4)	20.16	135.1	2.97	118.9	25.75	104.9	1.46	104.4	4.52
	0.25	4	94		120.7	11.00	112.2	29.23	133.3	2.22	118.9	23.83	105.3	1.47	103.6	4.92
	0.50	2	98		126.9	11.03	116.0	36.79	116.2	0.59	113.5	8.82	102.0	1.23	101.8	4.20
	0.75	2	66		115.5	11.02	110.7	23.77	112.6	0.61	108.5	8.64	100.6	1.14	100.6	1.78

Table 5b
Problems of Class II with n between 100 and 300. Average results over 5 instances, $\alpha_1 = 10$, $\alpha_2 = 1000$, $\mu = \frac{1}{5}n$; times are given in Digital VAXstation 3100 CPU seconds

					APCW		APCW + ref		AFJ		AFJ + ref		AV		AV + ref	
n	α	k	% ld	opt /lb	%sol	t	%sol	t	%sol	t	%sol	t	%sol	t	%sol	t
100	0.15	7	92	*	113.8 (3)	9.25	110.2 (3)	21.78	147.1	4.99	135.2	21.73	109.6 (4)	1.81	111.2	9.11
	0.20	5	93	*	121.9	9.31	115.4	26.64	134.7	1.06	119.4	27.36	105.9	1.72	104.6	7.33
	0.25	4	95		124.0 (4)	9.36	115.7 (4)	31.18	133.3	0.78	120.2	25.69	107.9	1.84	104.1	11.47
	0.50	2	98		128.7	9.39	120.0	34.61	118.8	0.66	114.3	13.47	101.5	1.83	100.3	4.45
	0.75	2	66		119.0	9.41	111.5	28.57	112.4	0.76	109.2	7.57	100.6	1.74	100.0	1.74
150	0.15	7	89	*	112.0	22.49	107.7	50.05	137.2	0.60	112.6	78.56	101.0	3.92	101.0	10.96
	0.20	5	95	*	115.5 (3)	22.52	109.3 (3)	68.51	134.8	0.65	118.6	70.53	100.9	3.98	100.7	11.01
	0.25	4	96		115.4 (2)	22.49	112.0 (2)	51.10	132.2	0.68	114.7	74.24	100.3	3.62	100.3	5.87
	0.50	2	99		112.0 (3)	22.52	106.7 (3)	58.58	118.1 (3)	1.09	109.3 (3)	48.81	100.2	4.41	100.0	6.77
	0.75	2	66		112.6	22.77	108.4	58.48	114.9 (1)	1.20	101.1 (1)	54.06	100.0	3.75	100.0	3.75
200	0.15	7	90	*	108.4 (4)	42.16	105.9 (4)	89.70	139.5	0.90	119.8	100.25	100.9	8.10	100.3	12.85
	0.20	5	96	*	111.1 (1)	41.89	103.2 (1)	147.29	138.0	1.02	114.6	150.53	100.6	8.15	100.0	8.15
	0.25	4	97		112.2 (4)	42.09	104.4 (4)	135.73	132.5	1.12	107.1	175.05	100.9	6.89	100.6	11.46
	0.50	2	99		111.4 (3)	42.10	106.3 (3)	118.75	116.0 (4)	1.85	105.8 (4)	82.47	100.6	9.37	100.0	9.37
	0.75	2	66		108.6	42.03	106.0	88.09	110.6 (4)	2.20	101.7 (4)	57.78	100.6	8.40	100.0	8.40
250	0.15	7	91	*	108.6	64.35	103.7	156.47	128.1	1.26	117.3	101.39	100.0	9.70	100.0	9.70
	0.20	5	97	*	107.9 (1)	63.96	105.3 (1)	138.97	125.5	1.44	112.0	119.68	100.6	9.61	100.6	17.64
	0.25	4	98		111.4 (1)	63.98	102.9 (1)	170.22	126.8	1.64	107.8	149.43	100.6	13.04	100.6	20.96
	0.50	2	99		112.1 (2)	64.13	103.6 (2)	217.95	115.5	2.75	104.6	107.02	100.0	12.07	100.0	12.07
	0.75	2	66		104.1	63.89	101.5	140.50	106.7	3.47	101.1	58.88	100.0	13.82	100.0	13.82
300	0.15	7	92	*	101.9 (3)	98.13	100.0 (3)	199.65	117.0	1.73	115.6	77.25	100.0	24.13	100.0	24.13
	0.20	5	97	*	– (0)	–	– (0)	–	117.3	2.00	106.1	129.33	100.0	20.35	100.0	20.35
	0.25	4	98		102.9 (1)	98.68	100.0 (1)	219.88	115.5	2.32	106.1	132.43	100.0	18.23	100.0	18.23
	0.50	2	99		106.4 (3)	98.42	100.0 (3)	256.38	110.0	3.85	102.0	113.85	100.0	35.78	100.0	35.78
	0.75	2	67		2295.9	137.79	1125.1	207.07	515.1	21.16	494.0	141.40	123.2	32.41	120.2	92.09

seen by noting that, over all the instances of Class I, the average percentage ratio of the solutions obtained by this algorithm without (resp. with) the refining step, is 153.5% (resp. 140.4%) with standard deviation 44.1 (resp. 31.9), while the solutions obtained by APCW have an average percentage ratio equal to 545.7% (resp. 352%), with standard deviation 520.5 (resp. 286.5), and the solutions produced by AFJ have an average percentage ratio equal to 391.1% (resp. 326.6%) with standard deviation 329.5 (resp. 259.7). These results are further illustrated by Fig. 6, which reports the trend of the percentage ratio of the solution found by the three algorithms with the refining step for $\alpha = 0.25$, and n between 10 and 100. The figure shows that the quality of the solutions found by APCW and AFJ algorithms quickly become worse as n increases, while the quality of the solution obtained by AV remains practically constant. Similar behaviour can be observed also for different α values, for larger instances, and without the refining step.

Algorithm AV proved also to be more efficient than APCW and AFJ. Indeed, over all the instances of Class I, the average computing time needed by AV without (resp. with) refining step is 5.2 (resp. 24.2) CPU seconds, while APCW requires on average 24.3 (resp. 78.1) seconds, and AFJ requires on average 8.6 (resp. 67.4) seconds. Fig. 7 reports the trend of the computing time needed by the three algorithms with the refining step, for $\alpha = 0.25$ and n between 10 and 100. Similar behaviour can also be observed for different values, for larger instances, and without the refining step.

These results also prove the good quality of the starting solutions that can be obtained with the additive procedures. The starting solutions can, in fact, be transformed into good feasible solutions with few insertions and exchanges. The refining step permits, in many cases, considerable closing of the gap existing between the optimal solution value (or a lower bound on its value) and the obtained approximate solution value. Note also that algorithm APCW was not able to find a feasible solution in 77 out of 350 instances (22% of the cases). The non-parametric version of the algorithm failed to obtain a feasible solution in 184 cases out of 350 (52.6%).

A second problem class has also been considered, whose instances are obtained from those of the previous class by 'triangularizing' the costs, i.e., by replacing each c_{ij} with the cost of the shortest path from i to j. The results obtained with the 350 instances of this second class are given in Tables 5a and 5b confirm what we already observed with the previous class. Indeed, even if a predictable improvement of the solution quality obtained by all the algorithms can be observed, in almost all the examined instances algorithm AV gives the best results, both with and without the refining step. Furthermore, in several cases the solution obtained by the AV algorithm is optimal or nearly so. Over all instances of Class II, the average percentage ratio of the solutions obtained by AV without (resp. with) refining step, is 104.6% (resp. 103.9%), while APCW and AFJ obtained solutions with an average percentage ratio equal to 113.2% (resp. 108.8%) and 122.3% (resp. 113.4%), respectively. The average computing time needed by APCW and AFJ is five to eight times higher than the time needed by AV.

Also for this problem class algorithm APCW is not able to find a feasible solution for 75 out of 350 instances (21.4%).

We also tackled real-world instances, coming from pharmaceutical and herbalist's product delivery in downtown Bologna. The relevant road network consists of 181 nodes, 92 bidirected links, and 224 directed links (these links come from one-way restrictions, which are massively imposed in downtown areas of most historical Italian cities). Each network node is defined by its coordinates in the plane, and corresponds to a relevant point in the city (junctions, or customers' locations). The cost of each link is computed as the Euclidean distance between its terminal vertices, rounded to the nearest integer (bidirected links are viewed as pairs of directed links).

There are 38 pharmacies in downtown Bologna, plus 32 herbalist's shops, each corresponding to a network node. All demands must be supplied by a single depot, located in the down-

Table 6
Real-world test problems for pharmaceutical product delivery in downtown Bologna. $\alpha_1 = 0$ or 10 (see text), $\alpha_2 = 1000$, $\mu = \frac{1}{5}n$; times are given in Digital VAXstation 3100 CPU seconds

n	*k*	% ld	APCW		APCW + ref		AFJ		AFJ + ref		AV		AV + ref	
			%sol	*t*	%sol	*t*	%sol	*t*	%sol	*t*	%sol	*t*	%sol	*t*
33	2	97	116.4	1.37	112.2	2.54	105.7	1.80	105.0	2.60	115.6	0.34	110.5	1.12
33	3	71	128.2	1.37	120.9	1.68	127.8	1.17	127.4	1.53	150.8	0.63	120.2	1.33
35	3	81	113.3	1.48	103.1	1.92	105.5	2.61	104.0	3.50	131.1	0.31	102.0	2.21
35	3	96	124.1	1.56	124.1	1.93	124.3	0.56	124.3	0.96	122.7	0.34	122.7	0.73
38	3	93	121.2	1.94	114.1	3.99	120.8	1.00	116.6	1.53	127.8	0.42	113.1	2.95
44	3	89	121.2	2.41	115.6	4.70	117.0	0.93	114.8	2.07	123.4	0.71	110.1	3.01
44	3	96	122.2	2.57	118.0	4.85	128.2	1.20	128.2	1.83	119.6	0.59	116.1	1.65
47	3	76	108.5	2.86	106.2	4.89	113.2	2.03	105.2	4.36	106.2	0.98	102.5	3.80
55	3	82	112.4	3.79	108.7	5.70	120.1	2.44	102.7	8.01	121.2	0.84	104.5	3.11
60	3	67	115.4	4.66	112.0	10.39	121.3	7.18	113.5	9.63	109.6	0.93	109.2	4.27
60	3	97	132.4	4.57	125.4	9.14	145.6	3.26	134.1	10.01	128.5	0.95	119.4	6.14
64	3	94	131.6	5.27	120.9	15.59	132.8	4.14	121.6	15.98	126.5	1.40	114.3	9.78
67	3	77	130.3	5.82	127.2	10.23	144.2	9.57	130.5	16.81	147.1	7.29	128.4	14.28
70	3	68	124.3	6.44	118.9	13.07	118.9	33.09	118.9	35.05	109.3	1.16	109.1	6.52
70	3	80	116.0	6.24	109.3	15.90	107.9	28.19	106.6	30.15	109.9	1.45	105.1	6.88
70	3	97	135.7	6.39	126.2	11.31	130.5	19.99	126.1	25.88	132.4	1.63	123.2	9.49

town area. Three vehicles are available, all with normalized capacity 1000.

Each problem instance corresponds to the requests of a subset S of the 70 customers, for a given day. The products have to be delivered in approximately 4 hours (from 8.30 a.m. to 12.30 a.m.), with no time-window restrictions. The only operative constraint is vehicle capacity, since 4 hours are always enough to complete the route. Given the demands, one sets up the associated ACVRP instances by constructing the $(|S|+1) \times (|S|+1)$ cost matrix (c_{ij}), where entries are computed as the shortest path in the network, from the node associated with i to that associated with j. The problem data are available, on request, from the author. Table 6 gives the computational results for 16 such instances, involving from 33 to 70 customers. We have used $\alpha_1 = 10$ for the most constrained instances (i.e. with average percentage load greater than 95%) while, for the remaining ones, we have used $\alpha_1 = 0$.

For this problem class, too, algorithm AV with the refining step has almost always obtained the best solution. Over all the real-world instances, the average percentage ratio of the solutions obtained by AV with the refining step is 113.2 while APCW and AFJ obtained an average percentage ratio equal to 116.4% and 117.5%, respectively. Moreover, the average computing time needed by APCW and AFJ are two or three times higher than the time needed by AV.

5. Conclusions

In this paper the extension to ACVRP of two well-known heuristic algorithms proposed in the literature for the symmetric case, has been discussed.

The asymmetric version of the Clarke–Wright algorithm preserves the quickness and easiness characteristics of the approach, but proved to be strongly ineffective, mainly because of the high number of solutions requiring more than k vehicles. An effective way to reduce this problem proved to be the use of a parametric technique for the computation of the saving. The value of the solutions obtained by the non-parametric and parametric Clarke–Wright algorithms quickly worsen as n increases.

The asymmetric version of the Fisher–Jaikumar algorithm shows a more homogeneous behaviour, particularly with triangularized instances. Nevertheless, the problem of the parameter choice for the seed selection procedure remains open. Indeed, the multistart approach used in the computational testing turns out, in practice, to be too time-consuming, requiring computing times 15 times higher than those reported in Section 4.

A new heuristic algorithm, called AV, especially tailored for ACVRP has also been described. The experimental testing illustrated in the previous Section shows the effectiveness of the proposed approach for the solution of both triangularized and non-triangularized instances. Indeed, the insertion procedure is able to build, in almost all tested instances, a feasible solution better than those that can be obtained with the other algorithms from the literature. This superiority is retained also after the refining step. The good quality of the initial solutions given by the additive bounding procedure is shown by the fact that these solutions are often 'almost' feasible. Moreover, the refining step, in many cases, is able to considerably close the gap between the optimal and the approximate solution values.

The AV algorithm has also been successfully used by Fischetti, Toth and Vigo [9] in a branch-and-bound algorithm for the exact solution of ACVRP. When applied to the solution of the additive bounding procedure at each node of the decision tree, algorithm AV has in many cases been able to determine the optimal solution within a few decision nodes, thus allowing for a considerable reduction of the overall computing time.

Possible extensions of the AV algorithm may consider an improvement of the refining step which can lead to further reduction of the gap between the optimal and the approximate solution. As an example, a non-deterministic criterion for the choice of the best exchange to be executed at each iteration can be considered. A multi-start approach and parametric techniques can also be considered since they have obtained good results with the Clarke–Wright and Fisher–Jaikumar algorithms.

Acknowledgements

This work has been supported by MURST 40%. The author gratefully thanks Professors Silvano Martello and Paolo Toth for their helpful comments.

References

[1] Bodin, L., Golden, B., Assad, A., and Ball, M., "Routing and scheduling of vehicles and crews. The state of the art", *Computers & Operations Research* 10 (1983) 63–211.

[2] Christofides, N., and Eilon, S., "An algorithm for the Vehicle Dispatching Problem", *Operations Research Quarterly* 20 (1969) 309–318.

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

[4] Christofides, N., Mingozzi, A., and Toth, P., "The Vehicle Routing Problem", in N. Christofides, A. Mingozzi, P. Toth and C. Sandi (eds.), *Combinatorial Optimization*, Wiley, Chichester, 1979.

[5] Christofies, N., Mingozzi, A., and Toth, P., "Exact algorithms for the Vehicle Routing Problem based on spanning tree and shortest path relaxations", *Mathematical Programming* 20 (1981) 255–282.

[6] Cournuejols, G., and Harche, F., "Polyhedral study of the Capacitated Vehicle Routing Problem", *Mathematical Programming* 60 (1993) 21–52.

[7] Fischetti, M., and Toth, P., "An additive bounding procedure for combinatorial optimization problems", *Operations Research* 37 (1989) 319–328.

[8] Fischetti, M., and Toth, P., "An additive bounding procedure for the Asymmetric Travelling Salesman Problem", *Mathematical Programming* 53 (1992) 173–197.

[9] Fischetti, M., Toth, P., and Vigo, D., "A branch and bound algorithm for the Capacitated Vehicle Routing Problem on directed graphs", *Operations Research* 42 (1994) 846–859.

[10] Fisher, M.L., and Jaikumar, R., "A generalized assignment heuristic for vehicle routing", *Networks* 11 (1981) 109–124.

[11] Fisher, M.L., "Optimal solution of Vehicle Routing Problems using minimum k-trees", *Operations Research* 42 (1994) 626–642.

[12] Laporte, G., Mercure, H., and Nobert, Y., "An exact algorithm for the Asymmetrical Capacitated Vehicle Routing Problem", *Networks* 16 (1986) 33–46.

[13] Laporte, G., "The Vehicle Routing Problem: An overview of exact and approximate algorithms", *European Journal of Operational Research* 59 (1992) 345–358.

[14] Lin, S., "Computer solutions to the Travelling Salesman Problem", *Bell System Technical Journal* 44 (1965) 2245–2269.

[15] Lin, S., and Kernighan, B.W., "An effective heuristic algorithm for the Travelling Salesman Problem", *Operations Research* 21 (1973) 498–516.

[16] Martello, S., and Toth, P., *Knapsack Problems: Algorithms and Computer Implementations*, Wiley, Chichester, 1990.

[17] Paessens, H., "The savings algorithm for the Vehicle Routing Problem", *European Journal of Operational Research* 34 (1988) 336–344.

[18] Savelsbergh, M.W.P., "Computer aided routing", Ph.D. Thesis, Centrum voor Wiskunde en Informatica, Amsterdam, 1988.