A Memetic Algorithm Solving the VRP, the CARP and General Routing Problems with Nodes, Edges and Arcs

Christian Prins and Samir Bouchenoua

LOSI, University of Technology of Troyes
BP 2060, 12 Rue Marie Curie
F-10010 Troyes Cedex, France
{Christian.Prins, Samir.Bouchenoua}@utt.fr

Summary. The VRP (Vehicle Routing Problem) and the CARP (Capacitated Arc Routing Problem) involve the routing of vehicles in an undirected network to service respectively a set of nodes or a set of arcs. Motivated by applications in waste collection, we define a more general model called NEARP (Node, Edge and Arc Routing Problem) for tackling mixed graphs with required nodes, edges and arcs. A memetic algorithm (MA) is developed for the NEARP. An evaluation on standard VRP and CARP benchmarks shows that the MA is competitive with most metaheuristics for these particular cases of the NEARP. We finally propose a set of NEARP instances, together with the solutions costs achieved by the MA, as a challenge for other researchers in vehicle routing.

Key words: memetic algorithm, vehicle routing, general routing problem.

1 Introduction

Traditionally, the literature devoted to multi-vehicle routing problems considers an undirected network and studies two distinct families of problems: *node routing problems* and *arc routing problems*, depending on the entities to be serviced in the network.

The *VRP* or *Vehicle Routing Problem* is a typical representative of node routing problems. It is usually defined on an undirected network in which some nodes correspond to customers. Each customer has a weight or demand for a commodity and a service cost. Each network edge has a travel cost. A fleet of identical vehicles of limited capacity is based at a depot node. A trip for a vehicle starts at the depot, visits a sequence of customers, and returns to the depot. The cost of a trip includes the service costs of its customers and the costs of each traversed edge.

The VRP consists of designing a set of trips of least total cost, such that each customer is visited exactly once and the total demand serviced by any trip does not exceed vehicle capacity. The VRP has important applications in logistics, for instance in distribution networks. It is unfortunately NP-hard and exact methods [1] have a limited interest, since some instances with 75 nodes (and even 50 nodes for the distance-constrained VRP) are not yet solved to optimality. This is why heuristics are required in practice for tackling real-life VRP instances. They comprise simple algorithms [2], like the merge heuristic from Clarke and Wright, and more recent and powerful metaheuristics like tabu search [3, 4, 5].

Comparatively, arc routing problems have been neglected for a long time by researchers, but they have raised a growing interest in the two last decades, mainly because of their applications like urban waste collection or winter gritting (see the good survey from Assad and Golden [6]). The problem corresponding to the VRP in arc routing is the *CARP* or *Capacitated Arc Routing Problem.* Its definition is similar but this time the tasks to be performed by the vehicles consist of servicing some edges, for instance spreading salt or collecting municipal refuse along a street.

The CARP is also NP-hard. Theoretically, it can be converted into an equivalent node routing problem as shown by Pearn et al. [7]. This transformation converts a CARP with k required arcs into a VRP with $3k+1$ nodes. Since the VRP itself is very hard, this increase in size is of course not acceptable and most researchers prefer to attack the CARP directly. The CARP seems more difficult than the VRP in practice: the exact solution methods published are still limited to small instances with at most 20 edges [8]. On the other hand, Belenguer and Benavent [9] have exploited the rich underlying structure of this problem to design an excellent lower bound, allowing an accurate evaluation of heuristics.

As for the VRP, the simplest heuristics published for the CARP are constructive methods, e.g. Path-Scanning from Golden et al. [10], Augment-Merge from Golden and Wong [11] and Ulusoy's tour splitting heuristic [12]. Metaheuristics have been designed more recently, like the powerful tabu search algorithm CARPET from Hertz, Laporte and Mittaz [13] and the genetic algorithms (GAs) from Lacomme, Prins and Ramdane-Chérif [14, 15]. The best of these GAs is the only algorithm able to reach the lower bound of Belenguer and Benavent [9] on 21 out of 23 standard instances proposed by DeArmon [16], containing up to 55 required edges.

Despite the success of metaheuristics for the VRP and the CARP, it is clear that these two problems cannot formalize the requirements of many real-world scenarios. Consider for instance urban waste collection. Although most tasks consist of servicing streets, the problem cannot be modeled as a pure CARP because of punctual accumulations of waste that must be modeled as required nodes (hospitals, schools, supermarkets, etc.). Moreover, an undirected graph can only model 2-way streets whose both sides are collected in parallel and in any direction (*zigzag* or *bilateral* collection, a practice reserved to low-traffic residential areas). In reality, a street can be a 2-way street with bilateral collection (giving an edge in the modeled network), a 2-way street with two sides collected independently (giving two opposite arcs), or even a 1-way street (giving one arc).

Our research is a step towards more generic models and algorithms able to handle such complications in vehicle routing. Section 2 presents our extended model, the *NEARP* or *Node, Edge and Arc Routing Problem.* It is defined on a mixed graph with required nodes, edges and arcs and contains the VRP and the CARP as particular cases. Section 3 describes three simple heuristics for the NEARP that are used to initialize the memetic algorithm (MA). The third one, a tour splitting method, plays also a key-role in chromosome evaluation. The MA itself is developed in section 4. It undergoes in section 5 a preliminary testing on standard VRP and CARP instances to check its competitiveness with respect to existing algorithms. A generator of instances for the new problem is described in section 6. We finally propose in section 7 a set of NEARP instances with the solution costs computed by the MA, as a challenge for OR researchers of the vehicle routing community. An appendix provides the reader with detailed tables of results and a list of formal definitions for all problems discussed.

2 The Node, Edge and Arc Routing Problem (NEARP)

This section formally defines the NEARP as a new problem generalizing both the VRP and the CARP and describes data structures for the algorithms of sections 3 and 4. The NEARP allows a mixed network with required nodes, edges and arcs. Contrary to the CARP, two distinct costs are handled for each link: one *deadheading cost*, i.e., the cost for a traversal without service (called *deadhead* by transporters) and one *service cost*, when the link is traversed to be treated. The entities to be serviced are directly tackled, i.e. the model does not rely on a conversion into a CARP or a VRP.

2.1 Problem statement

The NEARP is defined on a strongly connected and loopless mixed network $G = (N, E, A)$ with three sets of entities: a set N of n nodes, a set of edges E, and a set of arcs A. We call *links* the m entities in $E \cup A$. N includes a depot node s with a fleet of K identical vehicles of capacity W. The number of vehicles K is a decision variable. Each entity u has a non-negative traversal cost c_u. This cost is null for a node. For a link, it corresponds to a deadheading traversal (i.e., without service).

Some entities, the *tasks*, are *required*, i.e., they need to be processed by a vehicle. N_R, E_R and A_R respectively denote the subset of required nodes or *node-tasks*, the subset of required edges or *edge-tasks*, and the subset of required arcs or *arc-tasks*. Their cardinalities are respectively denoted by ν, ϵ and α. $\tau = \nu+\epsilon+\alpha$ denotes the total number of tasks. Each task $u = 1, 2, \ldots, \tau$ has a non-negative demand q_u and a non-negative processing cost p_u. To ensure feasibility, we assume that no demand exceeds W. Theoretically, all costs and demands should be integers, but our implementation accepts real numbers to handle some Euclidean instances from literature in section 5.

Any feasible vehicle trip must start from the depot, process a sequence of tasks whose total demand does not exceed W, and return to the depot. Its cost includes the processing costs of its tasks (required nodes, edges and arcs) and the traversal costs of the links used to travel from the depot to the first task, from each task to the subsequent one, and from the last task to the depot. The next subsection introduces data structures allowing to specify the cost of a trip by a concise formula.

Any feasible solution is a set of feasible trips covering all tasks. Tasks cannot be preempted, i.e., each task must appear in exactly one trip and only once in the sequence of tasks of that trip. Recall that the number of trips actually used, K, is not imposed but is part of the solution. The cost of a solution is the sum of its trip costs.

The NEARP consists of determining a least-cost solution. Clearly, this is a new problem that generalizes the VRP and the CARP: the VRP is the particular case with $A = \emptyset$ and $E_R = \emptyset$, while the CARP corresponds to $A = \emptyset$ and $N_R = \emptyset$. The *General Routing Problem* (*GRP*) is another special case of the NEARP, introduced by Orloff in 1974 [17]. In this generalization of the well-known *Traveling Salesman Problem* (*TSP*), one single vehicle must visit a subset of nodes and a subset of edges in an undirected graph to minimize the total mileage. Hence, the NEARP could also be called *Mixed Capacitated GRP* or *MCGRP*.

2.2 Internal network representation

Our algorithms rely on an internal network in which all entities (nodes and links) are encoded with the same attributes and stored in a common list L, indexed from 1 to $n + |A| + 2|E|$. The attributes for entity u are a begin node b_u, an end node e_u, a traversal cost c_u, a demand q_u, a processing cost p_u and a pointer $inv(u)$ explained below.

By convention, we set $b_u = e_u$ and $c_u = 0$ if entity u is a node: no confusion with a link is possible, since G is loopless. The required entities (tasks) are the ones with non-zero demands. Each required edge is encoded as two opposite arcs u and z linked thanks to their pointers inv, i.e., $e_u = b_z$, $e_z = b_u$, $inv(u) = z$ and $inv(z) = u$. These two arcs inherit their demands and their costs from the edge. Any arc or non-required edge u is such that $inv(u) = 0$. If u is a node, then $inv(u) = u$ by convention. Therefore, the three sets of tasks can be concisely defined by equations 1–3.

$$N_R = \{u \in L : b_u = e_u \wedge q_u > 0 \wedge inv(u) = u\} \tag{1}$$
$$E_R = \{u \in L : b_u \neq e_u \wedge q_u > 0 \wedge inv(u) \neq u\} \tag{2}$$
$$A_R = \{u \in L : b_u \neq e_u \wedge q_u > 0 \wedge inv(u) = 0\} \tag{3}$$

The costs of the shortest paths between any two entities can be pre-computed between their two end-nodes using Dijkstra's algorithm [18], resulting in a distance matrix D, $n \times n$. A trip θ is defined as a list $(\theta_1, \theta_2, \ldots, \theta_t)$ of task indexes, with a total demand $load(\theta) \leq W$ and a total cost $cost(\theta)$ defined by equations 4 and 5. Implicitly, θ starts and ends at the depot and shortest paths are assumed to connect the successive steps. A solution T is a list $(T_1, T_2, \ldots, T_K)$ of K vehicle trips (recall that K is a decision variable). Its cost is the sum of its trip costs. Each task appears exactly once in T and each edge-task occurs as one of its two opposite arcs.

$$load(\theta) = \sum_{i=1}^{t} q(\theta_i) \tag{4}$$

$$cost(\theta) = d(s, b(\theta_1)) + \sum_{i=1}^{t-1} (p(\theta_i) + d(e(\theta_i), b(\theta_{i+1}))) + p(\theta_t) + d(e(\theta_t), s) \tag{5}$$

3 Three simple heuristics for the NEARP

These heuristics are briefly described before the MA, because they are used to provide the initial population of the MA with good solutions. Moreover, the splitting technique of the third heuristic is also used in the MA for chromosome evaluation.

3.1 Nearest neighbor heuristic

Our *Nearest Neighbor Heuristic* or *NNH* adapts to the NEARP the Path-Scanning heuristic proposed by Golden and Wong for the CARP [10]. NNH is a sequential heuristic building the trips one by one until all tasks are processed. In building each trip, the sequence of tasks is extended at each iteration by joining the nearest free task z, until vehicle capacity W is exhausted. In NEARP instances with a majority of required links, the distance between the last task of the trip and the nearest free tasks is often zero, for instance when the tasks correspond to adjacent streets.

So, five rules are used to break ties among nearest tasks: 1) maximize the distance d_{zs} to the depot, 2) minimize this distance, 3) maximize a kind of yield q_z/p_z, 4) minimize this yield, 5) use rule 1 if the vehicle is less than half-full, else use rule 2. NNH computes one complete NEARP solution for each rule and returns the best one. A small example is given for rule 1 in Figure 1. Each black square represents a required node and each thick segment a required link. Thin lines correspond to shortest paths. The last task of the trip in construction is link u. The two nearest free tasks are node a and edge b, since $d_{ua} = d_{ub} = 3$. NNH will select edge b because $d_{bs} > d_{as}$.

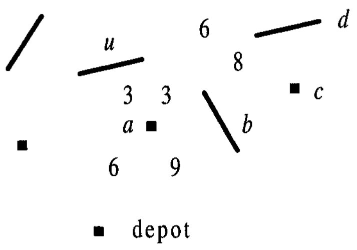

Fig. 1. Basic step of heuristic NNH with rule 1.

3.2 Merge heuristic

Our *Merge heuristic* or *MH* corresponds to the Clarke and Wright method for the VRP [2] and to the Augment-Merge heuristic for the CARP [11]. It starts with a trivial solution with τ trips reduced to one task. Then, each iteration evaluates the merger (concatenation) of any two trips, subject to W. For instance, in Figure 2, merging T_i and T_j yields a saving of $8+6-10=4$. MH merges the two trips with the largest positive savings. This process is repeated until no such merger is possible.

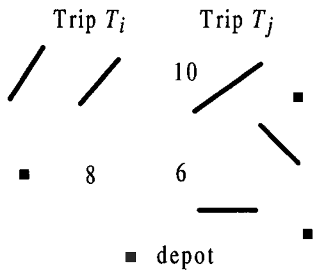

Fig. 2. Concatenation of two trips in the Merge Heuristic (MH).

Note that there exist up to 8 possible mergers for two trips T_i and T_j: one can put T_i before or after T_j and each trip may be inverted or not. In fact, the direction of each edge-task is changed in an inverted trip: e.g., if a trip contains a subsequence of two edge-tasks (u, z), then the inverted trip will contain the subsequence $(inv(z), inv(u))$. This also holds for a node u with $inv(u) = u$. Finally, the only case where a trip cannot be inverted is the presence of at least one arc-task u, since $inv(u) = 0$. In a real network, this occurs when a trip goes thru one-way streets.

3.3 Tour splitting heuristic

The *Tour Splitting Heuristic* or *TSH* extends a CARP algorithm from Ulusoy [12]. First, TSH relaxes vehicle capacity to build a giant tour S servicing all tasks. This can be done by any heuristic, for instance NNH called with $W = \infty$. Figure 3 shows such a giant tour $S = (a, b, c, d, e)$, with two node-tasks b and d and three required links a, c and e. The demand and processing cost of each task are given in brackets. An optimal procedure *Split* is then called to cut S into capacity-feasible trips.

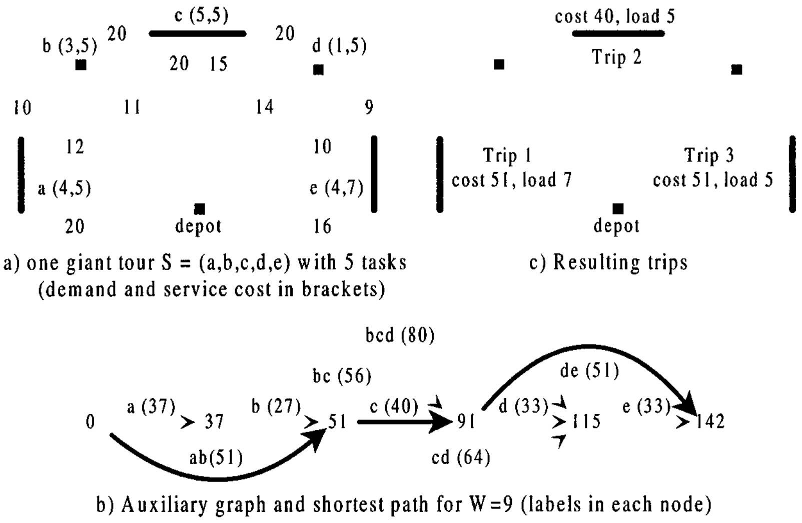

Fig. 3. Principle of the Tour Splitting Heuristic (TSH).

Split builds an auxiliary graph H with $\tau + 1$ nodes indexed from 0 to τ. Each subsequence $(S_i, S_{i+1}, \dots, S_j)$ of S that could give a capacity-feasible trip gives in H as one arc $(i - 1, j)$, weighted by the cost of the trip. This auxiliary graph is given in figure 3 for $W = 9$. Since H is acyclic by definition and contains $O(\tau^2)$ arcs, a shortest path from node 0 to node τ can be computed in $O(\tau^2)$ using Bellman's algorithm [18]. The resulting shortest path (boldface) indicates where to split the giant trip. It corresponds to a solution with 3 trips and a total cost equal to 142. Our implementation of TSH splits 5 giant trips, obtained by calling NNH with an infinite capacity and one priority rule at a time (see subsection 3.1). The best solution obtained is returned.

4 A memetic algorithm for the NEARP

4.1 Chromosomes and evaluation

A chromosome is simply defined as a sequence S of τ task indexes, *without trip delimiters.* It is almost a permutation chromosome because each task appears exactly once in S. However, each edge-task may appear as one of its two opposite arcs. Clearly, S does not directly represent a valid NEARP solution but it can be considered as a giant tour for a vehicle of infinite capacity. The *Split* procedure described in 3.3 for the TSH heuristic is used to extract from S the best possible NEARP solution. The guiding function $F(S)$ is nothing more than the cost of this solution. The following claim shows that the validity property stressed by Moscato in [19] holds. Hence, a memetic algorithm combining such chromosomes is expected to find an optimal NEARP solution.

Claim. The proposed chromosome structure is a *valid* representation.

Proof. By definition, *Split* converts any chromosome into an optimal NEARP solution (subject to the sequence order). Moreover, *there exists at least one optimal chromosome*: consider any optimal NEARP solution and concatenate its trips in any order. □

4.2 Extended OX crossover

Thanks to chromosomes without trip delimiters, classical crossovers for permutation chromosomes can be used for the NEARP. We quickly obtained good results by adapting the classical *Order Crossover* or *OX*, developed by Oliver et al. for the TSP [20]. This chromosome works well for cyclic permutations. Although a NEARP solution is not (strictly speaking) a permutation, it can be viewed as a cyclic list of trips because there is no reason to give a special role to a "first" or "last" trip.

Given two parents P_1 and P_2 of length τ, OX randomly draws two positions i and j with $1 \leq i \leq j \leq \tau$. To build the first child C_1, the substring $P_1(i) \dots P_1(j)$ is first copied into $C_1(i) \dots C_1(j)$. The tasks $P_2(j+1) \dots P2(\tau)$ and $P_2(1) \dots P_2(i-1)$ are then examined in that order. The tasks which are not yet present in C_1 are used to fill the empty slots of C_1, in the order $C_1(j+1) \dots C_1(\tau), C_1(1) \dots C_1(i-1)$.

```
Rank:  1   2   3   4   5   6   7   8   9
                  i=4     j=6
                   ↓       ↓
P1  :  1   3   2 | 6   4   5 | 9   7   8
P2  :  3   7   8 | 1   4   9 | 2   5   6

C1  :  8   1   9   6   4   5   2   3   7
C2  :  2   6   5   1   4   9   7   8   3
```

Fig. 4. Example of OX crossover

This process is illustrated by Figure 4. The other child C_2 is obtained in a similar way, by inverting the roles of P_1 and P_2. For the NEARP, the classical crossover must be adapted to take edge directions into account, i.e. a task u may be copied from P_1 to C_1 only if both u and $inv(u)$ are not already in the child. The extended crossover can be implemented in $O(\tau)$.

4.3 Local search procedure

To get a memetic algorithm, a *local search procedure* (*LSP*) replaces the mutation operator traditionally applied to new solutions created by recombination (children) after a crossover. Since LSP cannot work on chromosomes (without trip delimiters), the input chromosome S must be converted first into a NEARP solution, using the *Split* procedure of 3.3 LSP performs successive phases that scan in $O(\tau^2)$ the following types of moves, depicted in figures 5 and 6.

- Flip one task a, i.e., replace a by $inv(a)$ in its trip,
- Move one task a after another task or after the depot,
- Move two consecutive tasks a and b after another task or after the depot,
- Swap two tasks a and b,
- 2-opt moves depicted in figure 6.

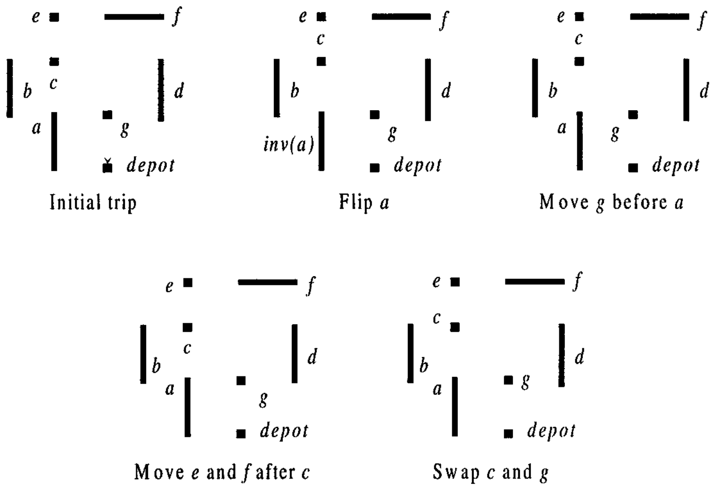

Fig. 5. Simple moves in the Local Search Procedure.

All these moves can be applied to one or two trips. Moreover, each task a moved to another location or swapped with another task may be inserted as a or $inv(a)$. For instance, the third move (move two tasks a and b) comprises in fact four distinct sub-cases: insert a and b, $inv(a)$ and b, a and $inv(b)$, or $inv(a)$ and $inv(b)$.

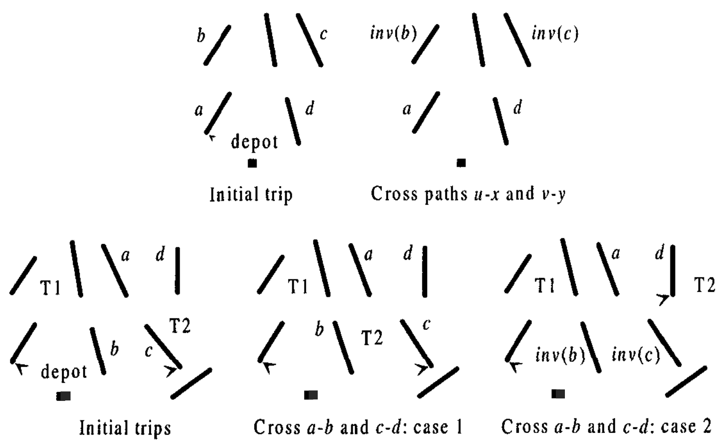

Fig. 6. 2-OPT moves on one trip and on two trips.

Each phase ends by performing the first improving move detected or when all moves have been examined. The loop on phases stops when a phase reports no improvement. The resulting NEARP solution is converted back into a chromosome by concatenating the tasks of its trips. In all cases, LSP terminates by applying *Split* to the result, because this sometimes decreases a bit the total cost.

On big instances, the neighborhood cardinality $O(\tau^2)$ leads to very time-consuming local searches, that typically absorb 95% of the total MA running time. To remedy this drawback, a classical neighbourhood reduction technique is used. We define for each task a list *neib*(a) that contains the τ tasks sorted in increasing order of distance to a and a threshold *thresh* between 1 and τ. Then, each iteration of the local search is restricted to all pairs (a, b), such that b belongs to the *thresh* first tasks in *neib*(a).

4.4 Population structure and initialization

The population is stored in an array Π of nc chromosomes, kept sorted in increasing order of costs (computed by *Split*). So, the best solution corresponds to Π_1. Identical solutions (*clones*) are forbidden to prevent a premature convergence of the MA (amplified by the local search) and to favour a better dispersal of solutions. Instead of an exact clone detection (e.g., using hashing methods), we adopt a simpler system in which the costs of any two solutions S_1, S_2 must be spaced at least by a constant $\Delta > 0$, i.e., $|F(S1) - F(S2)| \leq \Delta$. This condition is called the Δ-property. Its simplest form for integer costs is $\Delta = 1$, ensuring solutions with distinct costs.

At the beginning, the heuristics NNH, MH and TSH described in section 3 are executed. The local search procedure LSP of 4.3 is applied to the solutions computed by NNH and TSH, and after each merger for the Merge Heuristic MH. The resulting solutions are converted into chromosomes by concatenating their trips and stored in Π. The population is then completed by random chromosomes. On very small problems, it may be difficult to satisfy the Δ-property, especially if *nc* is large. In practice, we try up to *mnt* times to draw a random Π_k such that the Δ-property holds for $\Pi_1 \dots \Pi_k$. In case of failure, the number of chromosomes *nc* is truncated to $k - 1$.

Large populations raise another problem. During the MA, some crossovers are unproductive because their children violate the Δ-property and cannot be kept. The percentage of unproductive crossovers quickly increases with *nc* and with the local search rate. It is tolerable (less than 5%) if the population is relatively small (30-40 chromosomes) and if less than 20% of children undergo the local search.

Compared to the MA template proposed by Moscato [19], note that the local search is applied to the three initial heuristic solutions, but not to the random ones: because of the small population size, we are obliged to do so to have a sufficient dispersal of initial solutions and a better exploration of the solution space.

4.5 Basic iteration and stopping criteria

Each iteration of the MA starts by selecting two parents P_1 and P_2 by binary tournament: two chromosomes are randomly selected and the best one becomes P_1, this process is repeated to get P_2. The extended OX crossover (4.2) is applied to generate two children C_1 and C_2. One child C is selected at random, evaluated by *Split*, and improved by local search (4.3) with a fixed probability *pls*. An existing chromosome Π_k is drawn above the median cost ($k \geq nc/2$) to be replaced by C. The replacement is performed only iff the Δ-property holds for $(\Pi \setminus \{\Pi_k\}) \cup \{C\}$.

The MA stops after a maximum number of iterations *mni*, after a maximum number of crossovers without improving the best solution (Π_1) *mniwi*, or when a lower bound *LB* known for some instances is achieved.

4.6 Overall MA structure

The overall MA structure is given by Algorithm 1. The parameters are the population size *nc*, the minimal cost spacing Δ between any two solutions, the maximum number of tries *mnt* to get each initial random chromosome, the local search rate *pls*, the maximum number of iterations (crossovers) *mni*, the maximum number of iterations without improving the best solution *mnwi* and the lower bound *LB*.

```
Memetic Algorithm:
Begin
  run heuristics NNH, MH, TSH and improve solutions with LSP;
  discard solutions violating the $\Delta$-property;
  convert the remaining solutions into chromosomes, by concatenating their trips;
  $\Pi \leftarrow$ {resulting chromosomes};
  complete $\Pi$ with random chromosomes satisfying the $\Delta$-property;
  sort $\Pi$ in increasing cost order;
  $ni$, $niwi \leftarrow 0$;
  Repeat Until ( ($ni = mni$) or ($niwi = mniwi$) or ($F(\Pi_1) = LB$) ) Do
    $ni \leftarrow ni + 1$;
    select two parents $P_1$ and $P_2$ by binary tournament;
    apply OX to $P_1$, $P_2$ and choose one child $C$ at random;
    evaluate $C$ with $Split$;
    If ($random < pls$) Then
      improve $C$ with the local search procedure $LSP$;
    endIf
    draw $k$ at random between $\lfloor nc/2 \rfloor$ and $nc$ included;
    If ($\Pi \setminus \{\Pi_k\} \cup \{C\}$ satisfies the $\Delta$-property) Then
      $\Pi_k \leftarrow C$;
      If ($F(C) < F(\Pi_1)$) Then
        $niwi \leftarrow 0$;
        Else
        $niwi \leftarrow niwi + 1$;
      endIf
      shift $\Pi_k$ to keep $\Pi$ sorted;
    endIf
  endDo
End.
```

Fig. 7. Overall MA structure

5 Preliminary testing on VRP and CARP instances

5.1 Implementation and instances

The heuristics and the memetic algorithm have been programmed in the Pascal-like language Delphi version 5 and tested on a 1 GHz Pentium III PC with Windows 98. Before running the MA on NEARP instances, for which no published algorithm is available for comparison, we decided to test it on standard VRP and CARP instances.

The selected set of CARP instances (*gdb* files) contains 25 undirected problems built by DeArmon [16] and used by almost all algorithms published for the CARP. They can be downloaded on the Internet [21]. Instances 8 and 9 are discarded by all authors because they contain inconsistencies. The other files contain 7 to 27 nodes and 11 to 55 edges. All data are integers and all edges are required.

An excellent lower bound [9] is available for all these instances. The optimum is known for 21 instances out of 23, thanks to the tabu search CARPET of Hertz et al. [13] and the genetic algorithm of Lacomme et al. [15]. The only two remaining open instances are *gdb10* and *gdb14*. In spite of their relatively small size, the *gdb* instances are not so easy: for example, no constructive heuristic is able to solve more than two problems to optimality.

The set of VRP instances contains 14 Euclidean problems proposed by Christofides et al. [22]. They can be downloaded for instance from the OR Library [23]. They have 50 to 199 nodes. The network is complete and the costs are real numbers corresponding to the Euclidean distances between nodes. Files 6 to 10, 13 and 14 contain a route-length restriction. This constraint is easily handled by the MA, by ignoring the too long trips in the auxiliary graph built by the chromosome evaluation procedure *Split* (see 3.3).

The best-known solution costs to Christofides instances have been computed by various tabu search algorithms (TS) and simulated annealing procedures. They can be found for example in Gendreau et al. [3] and in Golden et al. [4]. As underlined by these authors, double-precision computations must be used to avoid cumulating rounding errors and to guarantee meaningful comparisons between final solution costs. No tight lower bound is available, but the best exact methods have proved that the solution values found for files 1 and 12 are in fact optimal.

5.2 Results for CARP instances

The MA parameters used for the *gdb* instances are $nc = 30$, $\Delta = 1$, $mnt = 60$, $pls = 0.1$, $mni = 20000$ and $mniwi = 6000$. Since these instances are not too large, the local search is set to a full aperture, i.e., $thresh = \tau$ (see 4.3).

Table 1 gathers the results for the CARP. The columns show, from left to right, the file name, the number of nodes n, the number of links m (equal to τ, since all edges are required), the best known solution value (BKS), the results obtained by the heuristics NNH, MH and TSH (followed by one call to the local search) and by the MA. The same setting of parameters is applied to all instances, except in the last column *Best MA* that reports the best solutions found with various settings during our experiments. The CPU time is given in seconds for all algorithms. The two last rows give the average deviation to the lower bound in % and the number of best solutions retrieved.

The MA solves 17 out of 23 instances to optimality, within reasonable CPU times (42 seconds on average, max. 4 minutes). The average deviation to the bound is quite small: 0.43%. To compare with, the best tabu search published [13] finds 18 optima, but with a slightly greater deviation of 0.48%. Using various settings, only two instances are improved (*gdb11* and *gdb24*).

5.3 Results for VRP instances

Table 2 reports the results found for the VRP in nearly the same format as table 1. However, the numbers of edges are here omitted because the networks are complete and, due to the lack of good lower bounds, the *Average* row now gives the average deviation to best-known solutions.

The MA parameters used this time are $nc = 30$, $\Delta = 0.5$, $mnt = 60$, $pls = 0.5$, $mni = 20000$ and $mniwi = 6000$. Neighborhood aperture is reduced to $thresh = 2 \times max\{10, \tau^{0.5}\}$. After the first phase with up to 20000 crossovers, the MA performs four short restarts of 2500 crossovers, in which the 7 worst chromosomes are replaced by random ones.

The MA finds 3 best-known solutions and the average deviation to best solutions is very small: 0.39%. The CPU time (10 minutes on average) exceeds 30 min only for one of the two largest instances with 199 nodes (*vrpnc10*, 42 min). In [4], Golden et al. list the results obtained by the 10 best TS methods for the VRP, which find 3 to 12 best-known solutions. Using several settings of parameters (*Best MA* column), the MA would be at rank 3 in this comparison, after three TS methods that respectively retrieve 12, 10 and 8 best-known solutions.

In a preliminary version of the MA, there was no neighborhood reduction technique in the local search and no restart. The average solution cost was only a bit larger, but the CPU time was excessive, exceeding 1 hour for four instances and reaching 2 hours and 53 minutes on *vrpnc10*.

The possibility of using a tabu search step for diversification has not been used for three reasons. Firstly, tabu search competitors are already available for the CARP and the VRP, so we wanted to develop in contrast a "pure" evolutionary algorithm. Secondly, a sufficient diversification seems to be provided by the restarts. Thirdly, we do not strictly follow Moscato's template and two features favour a good dispersal of solutions in the search space: a) the local search is not systematic and b) the population contains at any step distinct solutions.

6 Random generator of NEARP instances

A random generator has been designed to build NEARP instances. These networks are mixed, planar, strongly connected and imitate the shape of real street networks. The generation starts with a rectangle of basic squares. At the beginning, only the nodes at the corners of the basic squares exist, see a) in figure 8.

Four modifications can be applied to each square (see b) in figure 8): split vertically (V), horizontally (H), along the 1st diagonal (D1) and along the 2nd one (D2). Note that V and H create two new nodes and that a square may undergo up to four modifications. As from two, a central node is created to preserve planarity. We obtain in that way a planar undirected graph (see c) in figure 8). Since this graph is too regular, each node is randomly moved in a small circle. At this stage, provisional lengths in meters can be computed from node coordinates. To simulate streets that are a bit curved (without drawing them), a second perturbation consists of applying a random growth factor to each length (between 0 and 10% for instance).

Each edge is then converted into a one-way street with a given probability, by suppressing at random one of the two internal arcs that code the edge. Of course, strong connectivity is preserved. Traversal costs are computed from the length of each link, assuming an average deadheading speed of vehicles. Then, we decide for each link if its is required. If yes, we draw a non-zero demand at random.

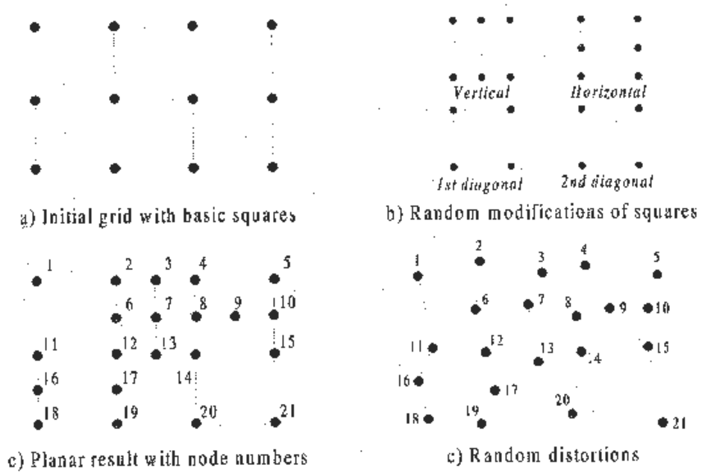
a) Initial grid with basic squares
Vertical
Horizontal
1st diagonal
2nd diagonal
b) Random modifications of squares
c) Planar result with node numbers
c) Random distortions
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

Fig. 8. Principles of random generation.

Finally, we decide for each 2-way street if it must be considered as one edge. If yes, the two arcs are linked with the *inv* pointer (see subsection 2.2) and the edge demand is the sum of quantities of the two sides. The processing cost of each task is computed as a function of its length, its demand, and a given vehicle processing speed. The instance generation ends by drawing vehicle capacity and depot location.

7 Selected set of NEARP instances with MA solutions

The generator has been used to build 23 large scale NEARP instances listed in table 3, with $n = 11 - 150$ nodes, $m = |A| + 2|E| = 29 - 311$ internal arcs and $\tau = 20 - 212$ tasks. The tasks comprise $\nu = 3 - 93$ node-tasks, $\epsilon = 0 - 94$

edge-tasks, and $\alpha = 0 - 149$ arc-tasks. All these files can be requested by an e-mail sent to the authors.

These networks are comparable in size to the ones observed in waste collection applications. Of course, the whole network of a big town can be much larger, but the collecting process is divided into sectors in practice. This defines an independent NEARP in each sector, with typically 100-200 street segments.

The MA parameters already applied to the VRP (including the restarts) are used, except $\Delta = 1$ instead of $\Delta = 0.5$, because all costs are integers in our NEARP instances. The results of the MA are listed in table 3. The average running time is 8 minutes of CPU time (max. 23 minutes). No published algorithm can be used for comparison and no good lower bound is available. This is why the table reports average deviations to the best MA solutions obtained by using various sets of parameters. However, by extrapolating the very good results achieved on the CARP and on the VRP, we think that the solutions values computed for the NEARP are quite good and other researchers are invited to try to obtain better results.

8 Conclusion

This paper presents a new problem, the NEARP, that generalizes the VRP and the CARP, and a memetic algorithm to solve it. Computational testing on standard VRP and CARP instances show that the MA can compete with the best metaheuristics published for these particular cases of the NEARP. Using a dedicated random network generator, we have built a set of 23 NEARP instances to evaluate the MA in the general case. The results are promising but the time spent in the local search procedure seems affected by the number of required nodes and should be improved by using more efficient neighborhoods for the instances with a majority of node-tasks. Beyond these interesting results, the main interest of this research is to solve several classical routing problems with one single algorithm.

References

1. Toth P, Vigo D (1998) Exact solution of the Vehicle Routing Problem. In: Crainic TG, Laporte G (eds) Fleet management and logistics, 1–31. Kluwer, Boston.
2. Laporte G, Gendreau M, Potvin JY, Semet F (2000) Classical and modern heuristics for the Vehicle Routing Problem. International Transactions in Operational Research 7:285–300.
3. Gendreau M, Laporte G, Potvin JY (1998) Metaheuristics for the Vehicle Routing problem. GERAD research report G-98-52, Montréal, Canada.

4. Golden BL, Wasil EA, Kelly JP, Chao IM (1998) The impact of metaheuristics on solving the Vehicle Routing Problem: algorithms, problem sets, and computational results. In: Crainic TG, Laporte G (eds) Fleet management and logistics, 33–56. Kluwer, Boston.
5. Toth P, Vigo D. The granular tabu search and its application to the Vehicle Routing Problem. To appear in INFORMS Journal on Computing.
6. Assad AA, Golden BL (1995) Arc routing methods and applications. In: Ball MO et al. (eds) Handbooks in OR and MS, volume 8, 375–483. Elsevier.
7. Pearn WL, Assad A, Golden BL (1987) Transforming arc routing into node routing problems. Computers and Operations Research 14:285–288.
8. Hirabayashi R, Saruwatari Y, Nishida N (1992) Tour construction algorithm for the Capacitated Arc Routing Problem. Asia-Pacific Journal of Operational Research 9:155–175.
9. Belenguer JM, Benavent E (2003) A cutting plane algorithm for the Capacitated Arc Routing Problem. Computers and Operations Research 30(5):705–728.
10. Golden BL, DeArmon JS, Baker EK (1983) Computational experiments with algorithms for a class of routing problems. Computers and Operations Research 10:47–59.
11. Golden BL, Wong RT (1981) Capacitated arc routing problems, Networks 11:305–315.
12. Ulusoy G (1985) The fleet size and mix problem for capacitated arc routing. European Journal of Operational Research 22:329–337.
13. Hertz A, Laporte G, Mittaz M (2000) A tabu search Heuristic for the Capacitated Arc Routing Problem. Operations Research 48:129–135.
14. Lacomme P, Prins C, Ramdane-Chérif W (2001) A genetic algorithm for the Capacitated Arc Routing Problem and its extensions. In: Boers EJW et al. (eds) Applications of evolutionnary computing. Lecture Notes in Computer Science 2037, 473–483. Springer, Berlin.
15. Lacomme P, Prins C, Ramdane-Chérif W. Competitive memetic algorithms for arc routing problems. To appear in Annals of Operations Research.
16. DeArmon JS (1981) A comparison of heuristics for the Capacitated Chinese Postman Problem. Master's thesis, The University of Maryland at College Park, MD, USA.
17. Orloff CS (1974) A fundamental problem in vehicle routing. Networks 4:35–64.
18. Cormen TH, Leiserson CL, Rivest ML, Stein C (2001) Introduction to algorithms, 2nd edition. The MIT Press, Cambridge, MA.
19. Moscato P (1999) Memetic algorithms: a short introduction. In: Corne D, Dorigo M and Glover F. (eds) New ideas in optimization, 219–234. McGraw-Hill.
20. Oliver IM, Smith DJ, Holland JRC (1987) A study of permutation crossover operators on the traveling salesman problem. In: Grefenstette JJ (ed) Proceedings of the 2nd Int. Conf. on Genetic Algorithms, 224–230. Lawrence Erlbaum, Hillsdale, NJ.
21. Belenguer JM, Benavent E. Directory with 3 sets of CARP instances. Web site: `http://www.uv.es/~belengue/carp.html`.
22. Christofides N, Mingozzi A, Toth P (1979) The Vehicle Routing Problem. In: Christofides N, Mingozzi A, Toth P, Sandi C (eds) Combinatorial optimization, 315–338. Wiley.

23. Beasley JE. Set of VRP instances from the OR library. Web site: `http://mscmga.ms.ic.ac.uk/jeb/orlib/vrpinfo.html`.
24. Crescenzi P, Kann V. A compendium of NP optimization problem. Web site: `http://www.nada.kth.se/~viggo/wwwcompendium/node103.html`.

Appendix

The problems studied in this paper can be described in the style used by Crescenzi and Kann [24] for their compendium of NP optimisation problems. The most general problem is the NEARP. Its highest-level particular cases are the CARP and the VRP. These three problems are not listed in the compendium.

Node, Edge and Arc routing problem (NEARP)

- INSTANCE: Mixed graph $G = (V, E, A)$, initial vertex $s \in V$, vehicle capacity $W \in IN$, subset $V_R \subseteq V$, subset $E_R \subseteq E$, subset $A_R \subseteq A$, traversal cost $c(u) \in IN$ for each "entity" $u \in V \cup E \cup A$, demand $q(u) \in IN$ and processing cost $p(u) \in IN$ for each required entity (task) $u \in V_R \cup E_R \cup A_R$.
- SOLUTION: A set of cycles (trips), each containing the initial vertex s, that may traverse each entity any number of times but process each task exactly once. The total demand processed by any trip cannot exceed W.
- MEASURE: The total cost of the trips, to be minimized. The cost of a trip comprises the processing costs of its serviced tasks and the traversal costs of the entities used for connecting these tasks.

Vehicle Routing Problem (VRP)

- INSTANCE: Complete undirected graph $G = (V, E)$, initial vertex $s \in V$, vehicle capacity $W \in IN$,length $c(e) \in IN$ for each $e \in E$, demand $q(i) \in IN$ for each $i \in V$.
- SOLUTION: A set of cycles (trips), each containing the initial vertex s, that collectively traverses every node at least once. A node must be serviced by one single trip and the total demand processed by any trip cannot exceed W.
- MEASURE: The total cost of the trips, to be minimized. The cost of a trip is the sum of its traversed edges.

Capacitated Arc Routing Problem (CARP)

- INSTANCE: Undirected graph $G = (V, E)$, initial vertex $s \in V$, vehicle capacity $W \in IN$, subset $E_R \subseteq E$, length $c(e) \in IN$ and demand $q(e) \in IN$ for each edge $e \in R$.

- SOLUTION: A set of cycles (trips), each containing the initial vertex s, that collectively traverse each edge of E_R at least once. Each edge of E_R must be serviced by one single trip and the total demand processed by any trip cannot exceed W.
- MEASURE: The total cost of the trips, to be minimized. The cost of a trip comprises the costs of its traversed edges, serviced or not.

However, the following special cases can be found in the compendium:

- the *minimum travelling salesperson*, an uncapacitated version of the VRP,
- the *minimum Chinese postman for mixed graphs* (an uncapacitated version of the CARP, but with a mixed network instead of an undirected one,
- the *minimum general routing problem*, which is an uncapacitated and undirected particular case of the NEARP.

Table 1. Computational results for CARP instances (see 5.2)

File	n	m	LB	BKS	NNH+LS	MH+LS	TSH+LS	MA	Time	BestMA
1	12	22	316	316*	345	323	316*	316*	< 0.01	316*
2	12	26	339	339*	345	359	352	339*	15.87	339*
3	12	22	275	275*	285	296	283	275*	1.10	275*
4	11	19	287	287*	287*	315	322	287*	< 0.01	287*
5	13	26	377	377*	395	395	383	377*	16.47	377*
6	12	22	298	298*	313	319	315	298*	1.81	298*
7	12	22	325	325*	346	325*	333	325*	< 0.01	325*
10	27	46	344	348	387	372	389	350	198.77	350
11	27	51	303	303*	339	329	346	311	115.12	309
12	12	25	275	275*	285	285	301	275*	1.21	275*
13	22	45	395	395*	430	427	412	395*	48.72	395*
14	13	23	450	458	498	474	520	458=	16.09	458=
15	10	28	536	536*	556	548	548	544	28.78	544
16	7	21	100	100*	100*	106	106	100*	< 0.01	100*
17	7	21	58	58*	58*	60	58*	58*	< 0.01	58*
18	8	28	127	127*	129	135	133	127*	8.24	127*
19	8	28	91	91*	91*	93	91*	91*	< 0.01	91*
20	9	36	164	164*	167	182	174	164*	0.88	164*
21	8	11	55	55*	55*	61	63	55*	< 0.01	55*
22	11	22	121	121*	123	125	125	121*	37.51	121*
23	11	33	156	156*	158	162	160	156*	7.36	156*
24	11	44	200	200*	205	205	207	202	235.96	201
25	11	55	233	233*	235	239	239	235	229.31	235
Average				0.13%	4.01%	5.47%	5.71%	0.43%	41.88s	0.36%
BKS retrieved					5	1	3	18		18

Times in seconds on a 1 GHz PC. Average deviations to LB in %.

Asterisks denote proven optima, '=' best-known solutions retrieved

Table 2. Computational results for VRP instances (see comments in 5.3)

File	n	BKS	NNH+LS	MH+LS	TSH+LS	MA	Time	BestMA
1	50	524.61*	578.84	548.58	566.70	524.93	62.40	524.61=
2	75	835.26	910.74	896.30	906.01	836.81	149.45	835.26=
3	100	826.14	863.23	872.52	867.40	829.72	300.71	827.39
4	150	1028.42	1138.94	1127.93	1062.56	1032.82	788.89	1032.82
5	199	1291.45	1356.11	1377.05	1371.20	1303.09	1698.62	1303.09
6	50	555.43	611.79	618.39	567.98	555.43=	112.32	555.43=
7	75	909.68	942.82	967.52	998.77	912.89	171.75	909.68=
8	100	865.94	924.65	910.23	898.75	866.87	320.22	865.94=
9	150	1162.55	1255.80	1280.49	1263.57	1169.69	1121.47	1169.69
10	199	1395.85	1530.77	1512.90	1524.74	1417.57	2539.26	1409.76
11	120	1042.11	1112.95	1052.81	1059.67	1045.55	342.41	1042.11=
12	100	819.56*	825.38	820.92	828.75	819.56*	1.70	819.56=
13	120	1541.14	1600.49	1573.23	1575.03	1548.58	924.61	1546.78
14	100	866.37	958.70	868.62	893.24	866.37=	284.18	866.37=
Average			7.13%	5.62%	5.20%	0.39%	629.86s	0.25%
BKS retrieved			0	0	0	3		8

Times in s on a 1 GHz PC. Average deviations to BKS in %.
Asterisks denote proven optima, '=' best-known solution retrieved.

Table 3. Computational results for the new NEARP instances (see section 7)

File	n	m	τ	ν	ϵ	α	NNH+LS	MH+LS	TSH+LS	MA	Time	BestMA
1	21	66	48	11	0	37	2853	2972	2878	2632	108.31	2589
2	68	246	185	36	0	149	13275	13195	13371	12336	1078.46	12241
3	31	96	79	16	8	55	4223	4164	4158	3702	157.04	3669
4	53	186	98	10	75	13	8121	8395	8436	7583	548.10	7583
5	32	65	65	23	4	38	5160	5048	5144	4562	100.02	4544
6	49	100	108	40	4	64	7760	7515	7826	7087	204.49	7087
7	75	166	168	54	8	106	10585	10536	10572	9974	662.62	9748
8	77	174	177	63	6	108	11685	11695	12112	10714	767.64	10658
9	29	86	50	6	39	5	4274	4274	4272	4041	140.83	4038
10	56	204	107	4	94	9	8190	8312	8248	7755	843.17	7582
11	69	241	82	65	6	11	5057	4716	4907	4503	414.68	4494
12	38	71	53	1	0	52	3429	3308	3532	3235	71.30	3235
13	150	294	141	79	2	60	10530	9756	10018	9339	550.57	9110
14	94	332	93	93	0	0	9296	8834	9106	8615	357.24	8566
15	52	182	91	0	91	0	9154	8785	9066	8359	390.19	8340
16	71	138	169	36	0	133	10205	9847	10498	9389	536.13	8933
17	42	134	63	16	16	31	4555	4775	4617	4165	116.12	4037
18	117	212	127	39	0	88	8001	8311	8087	7411	475.65	7254
19	126	311	212	61	9	142	17877	17605	18166	17036	1273.39	16554
20	43	133	73	38	2	33	5555	5127	5405	4918	164.56	4844
21	60	206	180	55	68	57	20023	19883	19702	18509	1370.61	18201
22	25	68	42	7	10	25	2187	2321	2222	1941	65.75	1941
23	11	29	20	3	2	15	784	780	820	780	20.38	780
Average							10.07%	8.87%	10.49%	1.26%	452.92s	0%
Solns of Best MA retrieved							0	1	0	5		23

Times in seconds on a 1 GHz PC. Average deviations to best MA taken as reference, in %.