Math. Program., Ser. B 105, 471–499 (2006)

Digital Object Identifier (DOI) 10.1007/s10107-005-0662-8

Roberto De Franceschi · Matteo Fischetti · Paolo Toth

# A new ILP-based refinement heuristic for Vehicle Routing Problems*

Received: May 23, 2004 / Accepted: December 1, 2004
Published online: 14 November 2005 – © Springer-Verlag 2005

**Abstract.** In this paper we address the Distance-Constrained Capacitated Vehicle Routing Problem (DCVRP), where $k$ minimum-cost routes through a central depot have to be constructed so as to cover all customers while satisfying, for each route, both a capacity and a total-distance-travelled limit.

Our starting point is the following refinement procedure proposed in 1981 by Sarvanov and Doroshko for the pure Travelling Salesman Problem (TSP): given a starting tour, (a) remove all the nodes in even position, thus leaving an equal number of "empty holes" in the tour; (b) optimally re-assign the removed nodes to the empty holes through the efficient solution of a min-sum assignment (weighted bipartite matching) problem. We first extend the Sarvanov-Doroshko method to DCVRP, and then generalize it. Our generalization involves a procedure to generate a large number of new sequences through the extracted nodes, as well as a more sophisticated ILP model for the reallocation of some of these sequences. An important feature of our method is that it does not rely on any specialized ILP code, as any general-purpose ILP solver can be used to solve the reallocation model.

We report computational results on a large set of capacitated VRP instances from the literature (with symmetric/asymmetric costs and with/without distance constraints), along with an analysis of the performance of the new method and of its features. Interestingly, in 13 cases the new method was able to improve the best-know solution available from the literature.

**Key words.** Vehicle Routing Problems – Heuristics – Large Neighborhood Search – Computational Analysis – Distance-Constrained Vehicle Routing Problem

## 1. Introduction

In this paper we address the following NP-hard (in the strong sense) *Distance-Constrained Capacitated Vehicle Routing Problem* (DCVRP). We are given a central depot and a set of $n-1$ customers, which are associated with the nodes of a complete undirected graph $G = (V, E)$ (where $|V| = n$ and node 1 represents the depot). Each edge $[i, j] \in E$ has an associated finite cost $c_{ij} \geq 0$. Each node $j \in V$ has a request $d_j \geq 0$ ($d_1 = 0$ for the depot node 1). Customers need to be served by $k$ cycles (*routes*) passing through the depot, where $k$ is fixed in advance. Each route must have a total duration (computed as the sum of the edge costs in the route) not exceeding a given limit

R. D. Franceschi: DEI, University of Padova, Via Gradenigo 6/A, 35131 Padova, Italy.
e-mail: robertodefranceschi@inwind.it

M. Fischetti: DEI, University of Padova, Via Gradenigo 6/A, 35131 Padova, Italy.
e-mail: matteo.fischetti@unipd.it

P. Toth: DEIS, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy.
e-mail: ptoth@deis.unibo.it

* Work supported by M.I.U.R. and by C.N.R., Italy.

$D$, and can visit a subset $S$ of customers whose total request $\sum_{j \in S} d_j$ does not exceed a given capacity $C$. The problem then consists of finding a feasible solution covering exactly once all the nodes $v \in V \setminus \{1\}$ and having a minimum overall cost; see, e.g., [7, 43].

We propose a new refinement heuristic for the DCVRP. The method is an elaboration of a refinement procedure originally proposed by Sarvanov and Doroshko [40] (SD) for the pure Travelling Salesman Problem (TSP), i.e., for the problem of finding a min-cost Hamiltonian cycle (*tour*) in a graph. (A similar methodology has been proposed, independently, by Gutin [26].) Given a starting TSP tour $T$ to improve, the SD procedure is based on two simple steps: (1) all the nodes in even position in $T$ are removed,[1] thus leaving an equal number of "empty holes" in the tour; (2) the removed nodes are optimally re-assigned to the empty holes through the efficient solution of a min-sum assignment (weighted bipartite matching) problem. An important property of the method is that it only requires polynomial time to implicitly enumerate an exponential number of alternative tours, i.e., it belongs to the family of Large Neighborhood Search (LNS) meta-heuristics (see, e.g., [1, 2, 4, 10–13, 25, 31]). As such, it has has been investigated theoretically by several authors, including Deineko e Woeginger [8], Weismantel [14], Punnen [36], Gutin [27] and Gutin, Yeo and Zverovitch [28]. We refer the reader to [5] for a thoughtful survey on recent VRP meta-heuristics.

Our approach goes far beyond the original SD scheme, and is based on a more sophisticated node removal policy followed by a procedure to construct a large number of new potential sequences through the extracted nodes. As a consequence, our reallocation cannot be rephrased as just a min-sum assignment problem, but it is based on the solution of a more sophisticated Integer Linear Programming (ILP) model. Our ILP has the structure of a set-partitioning model asking for the reallocation of a subset of the generated sequences, with the constraint that each extracted node has to belong to exactly one of the allocated sequences. Moreover, for each VRP route the new allocation has to fulfill the associated capacity and distance constraints. An important feature of our method is that it does not require a specialized ILP code, as any general-purpose ILP solver can be used to solve the allocation model.

As an extreme case, arising when extracting *all* the nodes and re-combining them in *all* possible sequences, our method then yields the well-known set partitioning VRP model (see, e.g., [43]). In this case, no matter the starting solution, the method would guarantee to return (in one step) a provably optimal solution, but it would require a typically unacceptable computing time for the construction of the sequences through the extracted nodes and for the exact solution of the associated set-partitioning ILP. At the other extreme, we have the Sarvanov-Doroshko approach where the nodes can only be extracted according to a rigid even-position criterion, only singleton node sequences are considered, and the ILP becomes essentially a min-sum assignment problem—plus the route constraints, in case the VRP instead of the TSP is addressed. The motivation of the present paper was precisely to find a proper balancing between the exact (yet too time consuming) set-partitioning model and the efficient (yet too rigid) Sarvanov-Doroshko proposal.

[1] Of course, the role of the even- and odd-position nodes could be interchanged

The paper is organized as follows. In Section 2 we describe the original method of Sarvanov and Doroshko. This basic method is extended in Section 3 and generalized to the DCVRP. Our generalization involves a procedure to generate a large number of new sequences through the extracted nodes, as well as a more sophisticated ILP model for the reallocation of some of these sequences. The implementation of the resulting SERR (for Selection, Extraction, Recombination, and Reallocation) algorithm is given in Section 4. Computational results on a large set of VRP instances from the literature (with symmetric/asymmetric costs and with/without distance constraints) are reported in Section 5, with an analysis of the performance of the method and of its positive features. Some solutions found by our method and improving the best-know solutions from the literature are finally illustrated in the Appendix.

## 2. The ASSIGN neighborhood for TSP

In their 1981 paper, Sarvanov and Doroshko [40] investigated the so-called ASSIGN neighborhood for the TSP, defined as follows: Given a certain TSP solution (viewed as node sequence $< v_1 = 1, v_2, \ldots, v_n >$), the neighborhood contains all the $\lfloor n/2 \rfloor !$ TSP solutions that can be obtained by permuting, in any possible way, the nodes in even position in the original sequence. In other words, any solution $(\psi_1, \psi_2, \ldots, \psi_n)$ in the neighborhood is such that $\psi_i = v_i$ for all odd $i$. An interesting feature of the neighborhood is that it can be explored exactly in polynomial time, though it contains an exponential number of solutions. Indeed, for any given starting solution the min-cost TSP solution in the corresponding ASSIGN neighborhood can be found efficiently by solving a min-cost assignment problem on a $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$ matrix; see e.g. [28]

Starting from a given solution, an improving heuristic then consists of exploring the Assign neighborhood according to the following two phases:

- *node extraction*, during which the nodes in even position (w.r.t. the current solution) are removed from the tour, thus leaving an equal number of "free holes" in the sequence;
- *node re-insertion*, during which the removed nodes are reallocated in the available holes in an optimal way by solving a min-sum assignment problem.

The simple example in Figure 1 gives an illustration of the kind of "improving moves" involved in the method. The figure draws a part of a tour, corresponding to the node sequence $\langle v_1, v_2, \ldots, v_9 \rangle$. In the node extraction phase, the nodes in even position $v_2$, $v_4$, $v_6$ e $v_8$ are removed from the sequence, whereas all the other nodes retain their position. In Figure 1b the black nodes represent the fixed ones, while the holes left by the extracted nodes are represented as white circles. If we use symbol "$-$" to represent a free hole, the sequence corresponding to Figure 1b is therefore $\langle v_1, -, v_3, -, v_5, -, v_7, -, v_9 \rangle$. The second step of the procedure, i.e., the optimal node reallocation, is illustrated in Figure 1c, where nodes $v_4$ and $v_6$ swap their position whereas $v_2$ and $v_8$ are reallocated as in the original sequence. This produces the improved part of tour $\langle v_1, v_2, v_3, v_6, v_5, v_4, v_7, v_8, v_9 \rangle$.

In the example, the same final tour could have been constructed by a simple 2-opt move. However, for more realistic cases the number of possible reallocation is exponential in the number of extracted nodes, hence the possible reallocation patterns are much

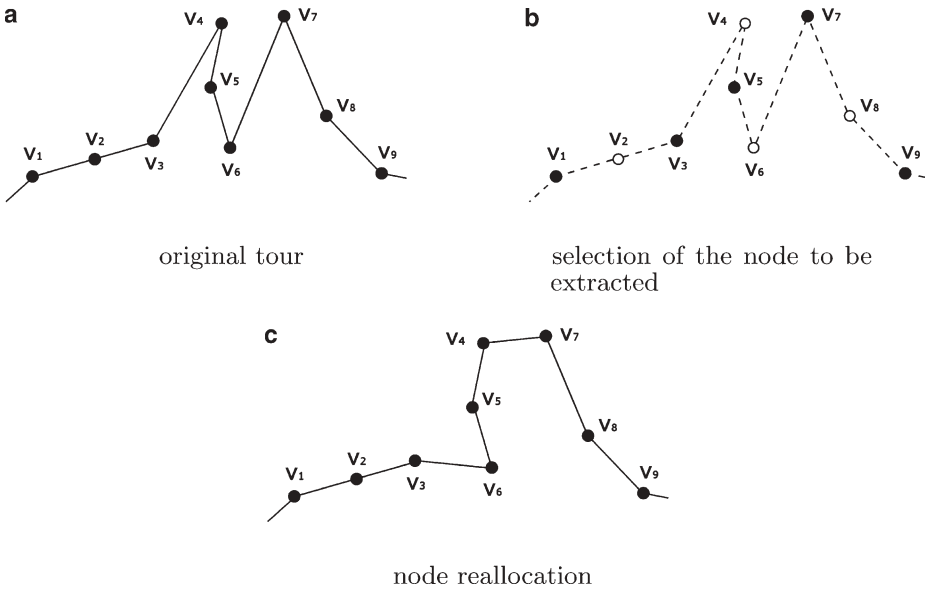


**Fig. 1.** A simple example of node extraction and reallocation

more complex and allow, e.g., for a controlled worsening of some parts of the solution which are compensated by large improvement in other parts.

Besides its important theoretical properties, the method suggests a framework for designing more and more sophisticated extract-and-reassign heuristics for TSP and related problems. A first step in this direction has been performed by Punnen [36], who suggested some variants of the basic method where node-sequences (as opposed to single nodes) are extracted and optimally reallocated by still solving a min-sum assignment problem. The (unpublished) preliminary results reported in [28], however, seem to suggest that this method is not very successful in practice. In our view, this is manly due to the too-rigid extract-and-reallocate paradigm, which is in turn a direct consequence of the requirement of using a min-cost assignment method to find an optimal reallocation. Our working hypothesis here was that much better results could be obtained in practice by replacing the (polynomially solvable) assignment problem by a more sophisticated set partitioning model (which is theoretically NP-hard, but effectively solvable in practice).

## 3. From TSP to DCVRP

We conjectured that the ASSIGN neighborhood would have been more useful in practice if applied to VRP problems rather than to the "pure" TSP. Indeed, due to the presence of several routes and of the associated route constraints, in VRP problems the node sequence is not the only issue to be considered when constructing a good solution: an equally-important aspect of the optimization is to find a balanced distribution of the nodes between the routes. In this respect, heuristic refinement procedures involving

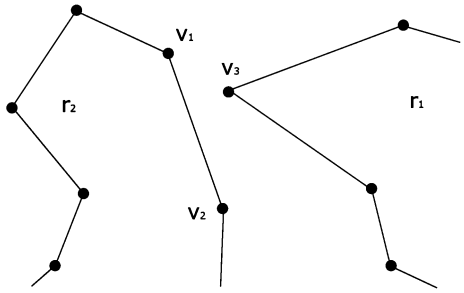


**Fig. 2.** The assignment of node $v_3$ to route $r_1$ is not optimal

complex patterns of node reallocations among the routes (akin to those in the span of the SD method) are likely to be quite effective in practice.

We therefore decided to extend the SD method to DCVRP so as to allow for more powerful move patterns, while generalizing its basic scheme so as to get rid off the too simple min-sum assignment model for node reallocation in favor of a more flexible reallocation model based on the (heuristic) solution of a more complex ILP model. The resulting method will be introduced, step by step, with the help of the examples reported in the sequel.

The first two very natural extensions we consider are akin to those proposed by Punnen [36]. Let us consider Figure 2, where a non-optimal part of a (geographical) VRP solution is depicted. It is clear that the position of node $v_3$ is not very clever, in that inserting $v_3$ between node $v_1$ and $v_2$ is likely to produce a better solution (assuming this new solution be feasible because of the route constraints). This move is however beyond the possibility of the pure SD method, where the extracted nodes can only be assigned to a hole left free by the removal of another node—while no hole between $v_1$ e $v_2$ exists which could accommodate $v_3$. The example then suggests a first extension of the basic SD method, consisting of removing the 1-1 correspondence between extracted nodes and empty holes. We therefore consider the concept of *insertion point*: after having extracted the selected nodes, we construct a *restricted solution* through the remaining nodes, obtained from the original one by short-cutting the removed nodes. All the edges in the restricted solution are then viewed as potential insertion points for the extracted nodes. In the example, removing $v_3$ but not $v_1$ and $v_2$ would produce the restricted solution depicted in Figure 3a, where all dashed edges are possible insertion points for the extracted nodes—this allows the method to produce the solution in Figure 3b.

A second important extension leads to a more flexible node extraction scheme that allows for the removal of a *sequence* of nodes, to be assigned as a whole to a certain insertion point. Note that, in case of symmetric costs, this assignment can be done in two different ways, depending on the orientation of the sequence. This however is not an issue in practice, as one can consider both orientations and keep the one producing the smallest reallocation cost.

So far we have assumed that a single insertion point can accommodate, at most, one of the extracted sequences. Unfortunately, it does not appear easy to get rid of this limitation. Instead, we suggest the use of a heuristic procedure to generate *new* sequences through the extracted nodes, to be allocated to the given insertion points.

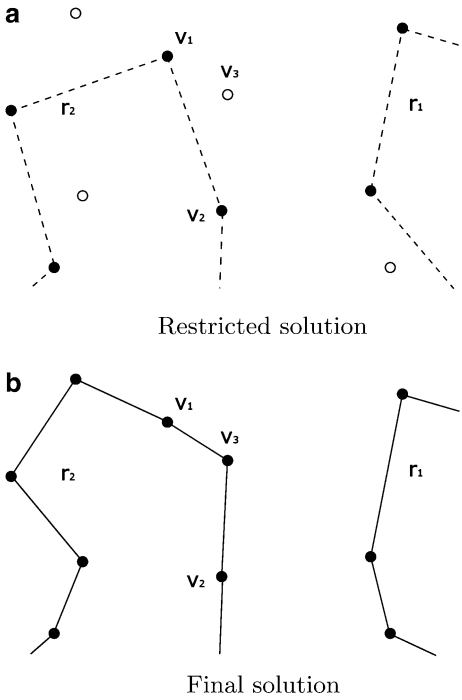


**Fig. 3.** Improving the solution depicted in Figure 2

To be more specific, starting from the extracted nodes sequences one can create new *derived sequences* that combine the extracted nodes in a different way. Of course, one never knows in advance which are the best sequences to be used, so all the (original and derived) sequences should be available when solving the reallocation problem.

The above considerations imply the use of a reallocation model which goes far beyond the scope of the original one, which is based on the solution of an easy min-cost assignment problem. Indeed, the new reallocation model becomes a set-partitioning ILP that receives as input the set of insertion points along with a (typically large) set of node sequences through the extracted nodes, and provides an (almost) optimal allocation of at most one sequence to each insertion point, with the constraint that each extracted node has to belong to one of the allocated sequences, while fulfilling the additional constraints on the capacity and distance constraints on the routes. This model will be described in more detail in the next section.

## 4. The SERR algorithm

Here is our specific implementation of the ideas outlined in the previous section, leading to the so-called *Selection, Extraction, Recombination, and Reallocation* (SERR) method.

(i) (*Initialization*). Apply a fast heuristic method to find a first (possibly infeasible, see below) DCVRP solution.

(ii) (*Selection*). Apply one of the available criteria (to be described later) to determine the nodes to be extracted—the nodes need not be consecutive, any node subset qualifies as a valid choice.

(iii) (*Extraction*). Extract the nodes selected in the previous step, and construct the corresponding restricted DCVRP solution obtained by short-cutting the extracted nodes. All edges in the restricted solution are put in the list $\mathcal{I}$ of the available insertion points.

(iv) (*Recombination*). The node sequences extracted in the previous step (called *basic* in the sequel) are initially stored in a *sequence pool*. Thereafter, heuristic procedures (to be described later) are applied to derive new sequences through the extracted nodes, which are added to the sequence pool. During this phase, dual information derived from the LP relaxation of the reallocation model can be used to find new profitable sequences—the so-called *pricing* step. Each sequence $s$ in the final pool is then associated with a (heuristically determined) subset $I_s$ of the available insertion points in $\mathcal{I}$. For all basic sequences $s$ we assume that $\mathcal{I}_s$ contains (among others) the *pivot* insertion point associated to $s$ in the original tour, so as to make it feasible to retrieve the original solution by just reallocating each basic sequence to the associated pivot insertion point.

(v) (*Reallocation*). A suitable ILP (to be described later in greater details) is set-up and solved heuristically through a general-purpose MIP solver. This model has a binary variable $x_{si}$ for each pair $(s, i)$, where $s$ is a sequence in the pool and $i \in \mathcal{I}_s$, whose value 1 means that $s$ has to be allocated to $i$. The constraints in the ILP stipulate that each extracted node has to be covered by exactly one of the selected sequences $s$, while each insertion point $i$ can be associated to at most one sequence. Further constraints impose the capacity and distance constraints in each route. Once an (almost) optimal ILP solution has been found, the corresponding solution is constructed and the current best solution is possibly updated (in which case each route in the new solution is processed by a 3-OPT [38] exchange heuristic in the attempt of further improving it).

(vi) (*Termination*). If at least one improved solution has been found in the last $n$ iterations, we repeat from step (ii); otherwise the method terminates.

### *4.1. Finding a starting solution*

Finding a DCVRP solution that is guaranteed to be feasible is an NP-hard problem, hence we have to content ourselves with the construction of solutions that, in some hard cases, may be infeasible—typically because the total-distance-travelled constraint is violated for some routes. In this case, the overall infeasibility of the starting solution can hopefully be driven to zero by a modification of the SERR recombination model where the capacity and distance constraints are treated in a soft way through the introduction of highly-penalized slack variables. (In addition, one could start with a partial solution, and use a modified SERR method to allocate the uncovered nodes.)

As customary in VRP problems, we assume that each node is assigned a coordinate pair $(x, y)$ giving the geographical position of the corresponding customer/depot in a 2-dimensional map.

One option for the initialization of the current solution required at step (i) of the SERR method, is to apply the classical two-phase method of Fisher and Jaikumar (FJ) [18]. This method can be implemented in a very natural way in our context in that it is based on a (heuristic) solution of an ILP whose structure is close to that of our reallocation model. In our implementation, the Fisher-Jaikumar heuristic is run $15 + n/k$ times with different choices of the seed nodes (spots) that give the initial shape of the routes.

According to our computational experience, however, the solution provided by the Fisher-Jaikumar heuristic is sometimes "too balanced", in the sense that the routes are filled so tightly that leave not enough freedom to the subsequent steps of our SERR procedure. Better results are sometimes obtained starting from a less-optimized solution whose costs exceed even by 20% the optimal cost, as e.g. the one obtained by using the following simple *sweep* method akin to the Gillett-Miller one [20]. We heuristically subdivide the customers in $k$ clusters according to the angle with the depot node, in such a way that the total request in each cluster hopefully does not exceed the vehicle capacity—if this is not the case, we first try to repair the cluster by means of simple node exchanges among the routes, and then resort to the Fisher-Jaikumar heuristic. Once the clusters have been determined, for each of them we construct a route by applying a TSP *nearest-neighbor* heuristic [38], followed by a 3-OPT phase—this typically produces a route that satisfies the total-distance constraint.

A second possibility is instead to start from an extremely-good solution provided by highly-effective (and time consuming) heuristics or meta-heuristics, in the attempt of improving this solution even further. In order to evaluate this option, in our computational experiments we considered the extreme case of starting from the best-known solution of some very hard instances, as reported in the literature.

### *4.2. Node selection criteria*

At each execution of step (ii) we apply one of the following selection schemes.

- scheme RANDOM-ALTERNATE: This criterion is akin to the SD one, and selects in some randomly-selected routes all the nodes in even position, while in the remaining routes the extracted nodes are those in odd position—the position parity being determined by visiting each route in a random direction.
- scheme SCATTERED: Each node had a uniform probability of 50% of being extracted; this scheme allows for the removal of consecutive nodes, i.e., of route subsequences.
- scheme NEIGHBORHOOD: Here we concentrate on a seed node, say $v^*$, and remove the nodes $v$ with a probability that is inversely proportional to the distance $c_{vv^*}$ of $v$ from $v^*$. To be more specific, once the seed node $v^*$ has been located (according to a criterion described in the sequel), we construct an ordered list containing the remaining nodes $v$ sorted by increasing distance $c_{vv^*}$ from $v^*$. Node $v^*$ is extracted along with the first 4 nodes in the list. The remaining part of the list is then subdivided into 5 sub-lists of equal size, and the corresponding nodes are extracted with a probability of 80%, 60%, 30%, 20%, and 10% for the first, second, third, fourth and fifth sub-list, respectively. The choice of the seed node $v^*$ is done with the aim of (a) improving the chances that the first seed nodes lead to a significant improvement of the incumbent solution, and (b) allowing each node to be selected with a certain

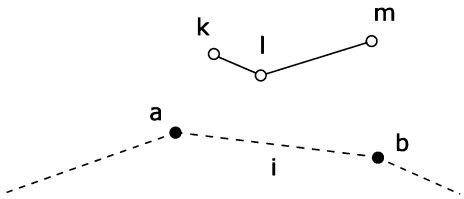


**Fig. 4.** Definition of the sequence cost and of the allowed insertion points

probability. This is obtained by assigning a score to each node $v^*$, which is defined so as to be proportional to the number of nodes which are "close" to $v^*$. A list of the potential seed nodes ordered by decreasing scores is then initialized. At each application of the NEIGHBORHOOD scheme, we select the next node in the list (in a circular way) to play the role of the seed node $v^*$.

Schemes RANDOM-ALTERNATE and SCATTERED appear particularly suited to improve the first solutions, whereas the NEIGHBORHOOD scheme seems more appropriate to deal with the solutions available after the first iterations. Therefore, in our SERR implementation, the above schemes are alternated as follows: we first apply 3 times the RANDOM-ALTERNATE scheme, then we apply 3 times the SCATTERED scheme, and afterwards the NEIGHBORHOOD scheme is used. Each time the incumbent solution is updated, the schemes are re-applied from the beginning: 3 times RANDOM-ALTERNATE, then 3 times SCATTERED, and thereafter NEIGHBORHOOD. This implies that in the first iterations of the method, where the incumbent solution is likely to be updated frequently, the RANDOM-ALTERNATE and SCATTERED scheme are mainly used. In the last iterations, on the other hand, the NEIGHBORHOOD scheme is mainly used.

### *4.3. Reallocation model*

Given the sequences stored in the pool and the associated insertion points (defined through the heuristics outlined in the next subsection), our aim is to reallocate the sequences so as to find a feasible solution of improved cost (if any). To this end, we need to introduce some additional notation, as illustrated in the example of Figure 4.

Let $\mathcal{F}$ denote the set of the extracted nodes, $\mathcal{S}$ the sequence pool, and $\mathcal{R}$ the set of routes $r$ in the restricted solution. For any sequence $s \in \mathcal{S}$, let $c(s)$ be the sum of the costs of the edges in the sequence, and let $d(s)$ be the sum of the requests $d_j$ associated with the internal nodes of $s$; e.g., $c(s) := c_{kl} + c_{lm}$ and $d(s) := d_l$ in the figure.

For each insertion point $i \in \mathcal{I}$ we then define extra-cost $\gamma_{si}$ for assigning sequence $s$ (in its best possible orientation) to the insertion point $i$; in the example, this cost is computed as $\gamma_{si} := c(s) + \min\{c_{ak} + c_{mb}, c_{am} + c_{kb}\} - c_{ab}$. For each route $r \in \mathcal{R}$ in the restricted solution, let $\mathcal{I}(r)$ denote the set of the insertion points (i.e., edges) associated with $r$, while let $\tilde{d}(r)$ and $\tilde{c}(r)$ denote, respectively, the total request and distance computed for route $r$—still in the restricted tour.

As already mentioned, our ILP model is based on the following decision variables.

$$x_{si} = \begin{cases} 1 \text{ if sequence } s \text{ is allocated to the insertion point } i \in \mathcal{I}_s \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

The model then reads:

$$\sum_{r\in\mathcal{R}} \tilde{c}(r) + \min \sum_{s\in\mathcal{S}} \sum_{i\in\mathcal{I}_s} \gamma_{si} x_{si} \tag{2}$$

subject to:

$$\sum_{s\ni v} \sum_{i\in\mathcal{I}_s} x_{si} = 1 \quad \forall v \in \mathcal{F} \tag{3}$$

$$\sum_{s\in\mathcal{S}: i\in\mathcal{I}_s} x_{si} \leq 1 \quad \forall i \in \mathcal{I} \tag{4}$$

$$\tilde{d}(r) + \sum_{s\in\mathcal{S}} \sum_{i\in\mathcal{I}_s\cap\mathcal{I}(r)} d(s) x_{si} \leq C \quad \forall r \in \mathcal{R} \tag{5}$$

$$\tilde{c}(r) + \sum_{s\in\mathcal{S}} \sum_{i\in\mathcal{I}_s\cap\mathcal{I}(r)} \gamma_{si} x_{si} \leq D \quad \forall s \in \mathcal{S},\ \ r \in \mathcal{R} \tag{6}$$

$$0 \leq x_{si} \leq 1 \text{ integer} \quad \forall s \in \mathcal{S},\ \ i \in \mathcal{I}_s \tag{7}$$

The objective function, to be minimized, gives the cost of the final DCVRP solution. Indeed, each coefficient gives the cost of an inserted sequence, including the linking cost, minus the cost of the "saved" edge in the restricted solution. Constraints (3) impose that each extracted node belongs to exactly one of the selected sequences, i.e., that it is covered exactly once in the final solution. Note that, in the case of triangular costs, one could replace $=$ by $\geq$ in (3), thus obtaining a typically easier-to-solve ILP having the structure of a set-covering (instead of set-partitioning) problem with side constraints. Constraints (4) avoid a same insertion point is used to allocated two or more sequences. Finally, constraints (5) and (6) impose that each route in the final solution fulfills the capacity and distance restriction, respectively.

In order to avoid to overload the model by an excessive number of variables, a particular attention has to be paid to reduce the number of sequences and, for each sequence, the number of the associated insertion points. As described in the next subsection, this is obtained by first generating them in a clever but conservative way, so as to produce a first set of variables for which the LP relaxation of the reallocation model can be solved easily. Afterwards, we enter a (*pricing*) loop where more and more (sequence, insertion point) pairs are generated. To this end, we first solve the LP relaxation of the current reallocation model, and save the corresponding optimal dual solution. For each candidate pair produced by our simple heuristics (to be discussed later), the LP reduced cost of the associated variable is computed by using the saved dual solution vector, and the pair is stored in case this cost is below a certain threshold.

As soon as the pricing loop does not produce any new variable, we freeze the current set of variables and invoke a general-purpose ILP solver to find an almost-optimal integer solution of the model. In our experiments, we used the commercial software ILOG

Cplex 8.0 with a limit of 30,000 branching nodes, emphasizing the search of integer solutions. Moreover, we provide on input to the ILP solver the feasible solution that corresponds to the current incumbent DCVRP solution, where each basic sequence is just reallocated to its corresponding pivot insertion point. In this way the ILP solver can immediately initialize its own incumbent solution, so every subsequent update (if any) will correspond to an improved DCVRP solution—the run being interrupted as soon as the internal ILP lower bound gives no hope to find such improvement.

### *4.4. Node recombination and construction of derived sequences*

This is a very crucial step in the SERR method, and is in fact one of the main novelties of our method. It consists not just of generating new "good" sequences through the extracted nodes, but also in associating each sequence to a clever set of possible insertion points that can conveniently accommodate it. Therefore, we have two complementary approaches to attack this problem: (a) we start from the insertion points, and for each insertion point we try to construct a reasonable number of new sequences which are likely to "fit well"; or (b) we start from the extracted nodes, and try to construct new sequences of small cost, no matter the position of the insertion points.

After extensive computational testing, we decided to implement the following two-phase method.

In the first phase, we initialize the sequence pool by means of the original (basic) sequences, and associate each of them to its corresponding (pivot) insertion point. This choice guarantees that the current DCVRP solution can be reconstructed by simply selecting all the basic sequences and putting them back in their pivot insertion point. Moreover, when the NEIGHBORHOOD selection scheme is used a further set of sequences is generated as follows. Let $v^*$ be the extracted seed node, and let $N(v^*)$ contain $v^*$ plus the 4 closest nodes (which have been extracted together with $v^*$). We apply a complete enumerative scheme to generate all the sequences through $N(v^*)$, of any cardinality, and add them to the pool. This choice is intended to increase the chances of improving locally the current solution, by exploiting appropriate sequences to reallocate the nodes in $N(v^*)$ in an optimal way.

The second phase is only applied for the NEIGHBORHOOD and SCATTERED selection schemes, and corresponds to a pricing loop based on the dual information available after having solved the LP relaxation of the current reallocation model.

At each iteration of the pricing loop, we consider, in turn, each insertion point $i \in \mathcal{I}$, and construct a number of new sequences that "fit well" with $i$. To be more specific, given the insertion point $i$ we apply the following steps.

(i) We initialize an iteration counter $L = 0$ along with a set $S$ containing a single dummy sequence $s =<>$ of cardinality 0, i.e., covering no nodes. At the generic iteration $L$, the set $S$ will contain at most $N_{max}$ (say) sequences of length $L$.

(ii) We set $L := L+1$ and generate new sequences of length $L$ according to the following scheme. For each $s =< v_1, v_2, \ldots, v_{|s|} > \in S$ and for each extracted node $v$ not in $s$, we generate all the sequences obtained from $s$ by inserting, in any possible way, node $v$ into $s$, namely sequences $< v, v_1, v_2, \ldots, v_{|s|} >$, $< v_1, v, v_2 \ldots, v_{|s|} >$, ..., $< v_1, v_2, \ldots, v, v_{|s|} >$, and $< v_1, v_2, \ldots, v_{|s|}, v >$.

(iii) For each sequence $s$ obtained at step (ii), we consider the variable $x_{si}$ associated with the allocation of $s$ to the given insertion point $i$, and evaluate the corresponding LP reduced cost $rc_{si}$.
(iv) We reset $S = \emptyset$, and then insert in $S$ the $N_{min}$ sequences $s$ with smallest $rc_{si}$, along with the sequences $s$ such that either $rc_{si} \leq \max\{RC_{max}, \delta\, rc^*\}$, where $rc^*$ is the smallest reduced cost generated so far, and $RC_{max}$ and $\delta$ are given parameters. In any case, no more than $N_{max}$ sequences are inserted. The corresponding variables $x_{si}$ are then added to the current LP model, but the LP is not re-optimized yet.
(vi) If the last sequences inserted in $S$ have a length smaller than $L_{max}$ (say), we repeat from step (ii); otherwise, the next insertion point $i$ is considered.

When the last insertion point has been considered, the current LP is re-optimized, and we repeat.

In our implementation, we defined $N_{min} = 5$ ($N_{min}^{(1)} = 10$ at the iteration $L = 1$), $N_{max} = 10$, $L_{max} = 5$, $\delta = 3$, and $RC_{max} = 10$.

In the very first application of steps (i)-(vi) above, when no LP has been solved yet, all dual LP variables are set to zero, and $RC_{max} = +\infty$. Moreover, when $L = 1$ (i.e., in case of singleton sequences $s =< v >$), at step (ii) we heuristically define $rc_{si} = c_{av} + c_{vb}$, where $[a, b]$ is the edge in the restricted solution that corresponds to insertion point $i$. This is because we noticed that the LP reduced costs tend to give an unreliable estimate of the real effectiveness of the variables associated with singleton sequences.

Still in the second phase, we construct the following additional sequences $s$ and add the corresponding variables $x_{si}$ to the LP model, but only for the insertion points $i \in \mathcal{I}$ (if any) such that the reduced cost $rc_{si} \leq RC_{max}$.

(i) all the sequences $s$ generated so far;
(ii) all possible sequences $s$ of cardinality 2 through the extracted nodes;
(iii) all possible sequences $s$ of cardinality 3 that can be obtained as an extension of the sequences $s'$ considered at step (ii), but only in case the corresponding variable $x_{s'i}$ has a reduced cost $rc_{s'i} \leq RC_{max}$ for at least one $i \in \mathcal{I}$.

Steps (i)–(iii) are iterated until no new variable is added to the model. Hashing techniques are used in order to avoid the generation of duplicated variables.

## 5. Computational results

Algorithm SERR has been tested on an AMD Athlon XP 2400+ PC with 1 GByte RAM. The ILP solver used in the experiments is ILOG Cplex 8.0 [29]. The algorithm has been coded in C++, and exploits the ILOG Concert Technology 1.2 interface [30]; the corresponding compiler is GNU gcc 3.0.4 with GNU GLIBC 2.2 libraries.

The performance of the algorithm was evaluated by considering two classes of experiments, corresponding to two different possibilities for finding the starting solution to be improved. In Class 1, the starting solution is obtained by means of one of the two fast initialization procedures described in Section 4.1 (namely, procedure FJ or procedure SWEEP), which are often quite far from optimality. In Class 2, instead, we start from an extremely-good feasible solution (typically, the best-known heuristic solution reported

in the literature), with the aim of evaluating the capability of our method to further improve it—this is of course possible only if the starting solution does not happen to be optimal, as it is likely the case for several of them.

Our computational analysis considers several Euclidean CVRP and DCVRP instances from the literature, that are generally used as a standard benchmark for the considered problems. Both *real cost* and *integer-rounded cost* instances have been considered. A group of CVRP instances with asymmetric costs (ACVRP) has been addressed as well, after a minor modification of our code needed to deal with directed (as opposed to undirected) graphs.

For the first class of experiments, CVRP, DCVRP and ACVRP instances with up to 100 customers were solved. In particular, our benchmark includes all the CVRP integer-rounded cost instances (with number of customers between 50 and 100) available at http://www.branchandcut.org/VRP, a site maintained by T. Ralphs (Lehigh University, Bethlehem, PA). These instances are denoted as X-nY-kZ, where "X", "Y", and "Z" represent, respectively, the instance "series" (i.e., A, B, E, P), the number $n$ of nodes of graph $G$, and the number $k$ of routes. In addition, we considered the integer-rounded cost instance F-n72-k4 as well as the eight classical VRP and DVRP real cost instances with no more than 100 customers described in Christofides and Eilon [6] and Christofides, Mingozzi and Toth [7]. These instances are denoted as WnY-Zu, where "W" is equal to "E" for the CVRP instances and "D" for the DVRP ones, while "Y" and "Z" have the meaning previously defined. For several instances, the optimal solution value has been found very recently by the exact algorithms proposed by Lysgaard, Letchford and Eglese [34], Fukasawa, Poggi de Aragão, Reis and Uchoa [19], Wenger [45], and Baldacci, Hadjiconstantinou and Mingozzi [3]. As to ACVRP, we considered the eight integer-rounded cost instances proposed (and solved to proven optimality) by Fischetti, Toth and Vigo [16]; these instances are denoted as A-nY-kZ.

Tables 1 and 2 report the results obtained by algorithm SERR with initialization procedures FJ and SWEEP, respectively, when applied to CVRP instances. A time limit of 7,200 seconds (i.e., 2 hours) was imposed for each run. All the computing times reported in the tables are expressed in AMD Athlon XP 2400+ CPU seconds. The columns in the tables have the following meaning:

- **Best** is the best known solution value from the literature (including the improvements found in [19, 45, 3]; provable optimal values are marked by $^*$);
- **Start** is the value of the initial solution;
- **SERR** is the value of the solution found by SERR;
- **%err** is the percentage error between the value found by SERR and the best solution value;
- **%imp** is the percentage improvement of the value found by SERR with respect to the initial solution value;
- **Total** is the total computing time required by SERR to obtain its final solution;
- **Cplex** is the computing time required by Cplex to solve the ILP instances;
- **Recomb.** is the computing time required by SERR for the *Recombination* phase.

The last three columns of the two tables refer a simplified version of SERR in which the recombination phase (see Section 4.4) is not performed. Columns have the following meaning:

**Table 1.** Complete method and no recombination method, with FJ initial solution. (*) provable optimal solution value

| | | Solution values | | | | Computing times | | | No recombination | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Problem | Best | Start | SERR | %err | %imp | Total | Cplex | Recomb. | Sol. | %imp | Time |
| A-n53-k7 | * 1010 | 1092 | 1011 | 0.1 | 7.4 | 141.0 | 16.5 | 89.7 | 1021 | 6.5 | 3.1 |
| A-n54-k7 | * 1167 | 1230 | 1179 | 1.0 | 4.1 | 82.3 | 4.6 | 61.0 | 1195 | 2.8 | 0.3 |
| A-n55-k9 | * 1073 | 1159 | 1073 | 0.0 | 7.4 | 171.6 | 14.9 | 121.7 | 1125 | 2.9 | 1.6 |
| A-n60-k9 | * 1354 | 1448 | 1363 | 0.7 | 5.9 | 670.6 | 307.1 | 287.1 | 1363 | 5.9 | 0.2 |
| A-n61-k9 | * 1034 | 1133 | 1064 | 2.9 | 6.1 | 271.3 | 120.9 | 107.4 | 1094 | 3.4 | 2.5 |
| A-n62-k8 | * 1288 | 1388 | 1288 | 0.0 | 7.2 | 356.5 | 14.9 | 296.1 | 1301 | 6.3 | 9.9 |
| A-n63-k9 | * 1616 | 1700 | 1641 | 1.5 | 3.5 | 78.1 | 21.8 | 44.5 | 1651 | 2.9 | 13.1 |
| A-n63-k10 | * 1314 | 1441 | 1319 | 0.4 | 8.5 | 203.0 | 40.5 | 127.8 | 1323 | 8.2 | 7.6 |
| A-n64-k9 | * 1401 | 1550 | 1431 | 2.1 | 7.7 | 565.0 | 236.5 | 268.7 | 1407 | 9.2 | 16.7 |
| A-n65-k9 | * 1174 | 1203 | 1174 | 0.0 | 2.4 | 123.2 | 0.8 | 92.2 | 1190 | 1.1 | 0.6 |
| A-n69-k9 | * 1159 | 1216 | 1159 | 0.0 | 4.7 | 207.8 | 20.2 | 163.1 | 1180 | 3.0 | 0.4 |
| A-n80-k10 | * 1763 | 1862 | 1793 | 1.7 | 3.7 | 2087.7 | 745.3 | 1201.9 | 1804 | 3.1 | 41.2 |
| B-n50-k7 | * 741 | 764 | 743 | 0.3 | 2.7 | 248.2 | 90.0 | 120.8 | 741 | 3.0 | 0.5 |
| B-n50-k8 | * 1312 | 1377 | 1324 | 0.9 | 3.8 | 2233.2 | 1444.5 | 722.7 | 1331 | 3.3 | 5.5 |
| B-n51-k7 | * 1032 | 1056 | 1032 | 0.0 | 2.3 | 561.8 | 246.9 | 257.7 | 1032 | 2.3 | 5.1 |
| B-n52-k7 | * 747 | 759 | 747 | 0.0 | 1.6 | 428.2 | 299.4 | 108.9 | 750 | 1.2 | 5.0 |
| B-n56-k7 | * 707 | 743 | 710 | 0.4 | 4.4 | 370.0 | 160.5 | 191.9 | 726 | 2.3 | 1.4 |
| B-n57-k7 | * 1153 | 1186 | 1153 | 0.0 | 2.8 | 662.5 | 355.6 | 266.8 | 1172 | 1.2 | 1.1 |
| B-n57-k9 | * 1598 | 1667 | 1625 | 1.7 | 2.5 | 1269.0 | 547.3 | 639.4 | 1645 | 1.3 | 0.6 |
| B-n63-k10 | * 1496 | 1581 | 1548 | 3.5 | 2.1 | 178.4 | 63.4 | 97.3 | 1554 | 1.7 | 0.4 |
| B-n64-k9 | * 861 | 913 | 863 | 0.2 | 5.5 | 1262.7 | 681.1 | 532.7 | 890 | 2.5 | 3.7 |
| B-n66-k9 | * 1316 | 1385 | 1322 | 0.5 | 4.5 | 4055.3 | 1211.2 | 2748.8 | 1326 | 4.3 | 19.8 |
| B-n67-k10 | * 1032 | 1075 | 1033 | 0.1 | 3.9 | 1826.6 | 387.4 | 1358.6 | 1065 | 0.9 | 4.6 |
| B-n68-k9 | * 1272 | 1326 | 1288 | 1.3 | 2.9 | 437.3 | 295.5 | 128.2 | 1288 | 2.9 | 13.7 |
| B-n78-k10 | * 1221 | 1278 | 1231 | 0.8 | 3.7 | 337.4 | 43.3 | 276.7 | 1251 | 2.1 | 19.9 |

**Table 1.** (cond.)

| | | Solution values | | | | Computing times | | | No recombination | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Problem | Best | Start | SERR | %err | %imp | Total | Cplex | Recomb. | Sol. | %imp | Time |
| E-n51-k5 | * 521 | 521 | 521 | 0.0 | 0.0 | 0.7 | 0.0 | 0.0 | 521 | 0.0 | 0.0 |
| E-n76-k7 | * 682 | 705 | 697 | 2.2 | 1.1 | 99.6 | 0.0 | 95.3 | 702 | 0.4 | 0.5 |
| E-n76-k8 | * 735 | 751 | 737 | 0.3 | 1.9 | 670.6 | 0.8 | 638.6 | 740 | 1.5 | 20.4 |
| E-n76-k10 | * 830 | 848 | 840 | 1.2 | 0.9 | 202.9 | 11.4 | 180.2 | 843 | 0.6 | 8.7 |
| E-n76-k14 | * 1021 | 1075 | 1027 | 0.6 | 4.5 | 3382.5 | 1713.5 | 1540.3 | 1059 | 1.5 | 18.6 |
| E-n101-k8 | * 815 | 834 | 822 | 0.9 | 1.4 | 5641.1 | 1.7 | 5544.0 | 831 | 0.4 | 5.9 |
| E-n101-k14 | 1071 | 1153 | 1088 | 1.6 | 5.6 | 5094.2 | 2021.6 | 3021.3 | 1103 | 4.3 | 43.0 |
| P-n50-k8 | * 631 | 649 | 631 | 0.0 | 2.8 | 124.7 | 21.8 | 71.1 | 649 | 0.0 | 0.0 |
| P-n55-k10 | * 694 | 718 | 704 | 1.4 | 1.9 | 100.0 | 8.2 | 78.4 | 712 | 0.8 | 4.8 |
| P-n60-k10 | * 744 | 771 | 747 | 0.4 | 3.1 | 123.8 | 8.7 | 102.5 | 758 | 1.7 | 7.9 |
| P-n60-k15 | * 968 | 1024 | 974 | 0.6 | 4.9 | 778.0 | 239.1 | 453.1 | 992 | 3.1 | 2.0 |
| P-n65-k10 | * 792 | 822 | 802 | 1.3 | 2.4 | 830.5 | 60.4 | 699.2 | 818 | 0.5 | 1.4 |
| P-n70-k10 | * 827 | 843 | 836 | 1.1 | 0.8 | 181.8 | 15.4 | 153.6 | 842 | 0.1 | 2.2 |
| P-n101-k4 | * 681 | 703 | 686 | 0.7 | 2.4 | 558.2 | 0.2 | 542.4 | 689 | 2.0 | 15.1 |
| F-n72-k4 | * 237 | 245 | 237 | 0.0 | 3.3 | 363.1 | 5.4 | 351.8 | 245 | 0.0 | 0.0 |
| E051-05e | * 524.61 | 524.61 | 524.61 | 0.0 | 0.0 | 0.8 | 0.0 | 0.0 | 524.61 | 0.0 | 0.0 |
| E076-10e | 835.26 | 855.91 | 845.84 | 1.3 | 1.2 | 1588.9 | 181.8 | 1319.7 | 849.68 | 0.7 | 17.7 |
| E101-08e | 826.14 | 849.29 | 834.75 | 1.0 | 1.7 | 6200.2 | 7.9 | 6069.9 | 835.75 | 1.6 | 39.8 |
| E101-10c | 819.56 | 825.65 | 819.56 | 0.0 | 0.7 | 2923.6 | 0.9 | 2873.2 | 819.56 | 0.7 | 18.2 |

**Table 2.** Complete method and no recombination method, with SWEEP initial solution. (*) provable optimal solution value

| | | Solution values | | | | Computing times | | | No recombination | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Problem | Best | Start | SERR | %err | %imp | Total | Cplex | Recomb. | Sol. | %imp | Time |
| A-n53-k7 | * 1010 | 1295 | 1017 | 0.7 | 21.5 | 222.4 | 14.0 | 156.0 | 1017 | 21.5 | 10.7 |
| A-n54-k7 | * 1167 | 1363 | 1172 | 0.4 | 14.0 | 208.3 | 34.2 | 138.9 | 1201 | 11.9 | 7.4 |
| A-n55-k9 | * 1073 | 1254 | 1073 | 0.0 | 14.4 | 108.0 | 18.4 | 69.3 | 1081 | 13.8 | 9.2 |
| A-n60-k9 | * 1354 | 1740 | 1358 | 0.3 | 22.0 | 308.5 | 72.5 | 204.0 | 1403 | 19.4 | 16.3 |
| A-n61-k9 | * 1034 | 1334 | 1038 | 0.4 | 22.2 | 1539.2 | 839.6 | 577.3 | 1113 | 16.6 | 6.3 |
| A-n62-k8 | * 1288 | 1656 | 1288 | 0.0 | 22.2 | 981.2 | 216.0 | 667.8 | 1321 | 20.2 | 10.1 |
| A-n63-k9 | * 1616 | 2149 | 1627 | 0.7 | 24.3 | 837.2 | 457.7 | 326.2 | 1662 | 22.7 | 15.4 |
| A-n63-k10 | * 1314 | 1943 | 1322 | 0.6 | 32.0 | 1665.6 | 778.1 | 758.4 | 1332 | 31.4 | 21.4 |
| A-n64-k9 | * 1401 | 1786 | 1410 | 0.6 | 21.1 | 928.8 | 462.2 | 404.2 | 1430 | 19.9 | 12.2 |
| A-n65-k9 | * 1174 | 1577 | 1177 | 0.3 | 25.4 | 849.9 | 466.5 | 319.8 | 1230 | 22.0 | 11.0 |
| A-n69-k9 | * 1159 | 1491 | 1163 | 0.4 | 22.0 | 344.8 | 54.1 | 255.3 | 1199 | 19.6 | 9.2 |
| A-n80-k10 | * 1763 | 2215 | 1780 | 1.0 | 19.6 | 2912.4 | 1563.7 | 1251.6 | 1786 | 19.4 | 16.5 |
| B-n50-k7 | * 741 | 1082 | 741 | 0.0 | 31.5 | 24.5 | 7.9 | 13.5 | 741 | 31.5 | 13.6 |
| B-n50-k8 | * 1312 | 1458 | 1318 | 0.5 | 9.6 | 3637.2 | 2624.0 | 952.6 | 1322 | 9.3 | 6.1 |
| B-n51-k7 | * 1032 | 1168 | 1032 | 0.0 | 11.6 | 477.7 | 267.2 | 172.5 | 1057 | 9.5 | 5.3 |
| B-n52-k7 | * 747 | 1005 | 747 | 0.0 | 25.7 | 196.0 | 104.3 | 78.3 | 747 | 25.7 | 19.5 |
| B-n56-k7 | * 707 | 941 | 710 | 0.4 | 24.5 | 696.0 | 295.7 | 366.5 | 748 | 20.5 | 8.6 |
| B-n57-k7 | * 1153 | 1701 | 1193 | 3.5 | 29.9 | 5968.0 | 4182.5 | 1692.9 | 1236 | 27.3 | 24.1 |
| B-n57-k9 | * 1598 | 2126 | 1599 | 0.1 | 24.8 | 1500.9 | 631.1 | 781.4 | 1614 | 24.1 | 16.1 |
| B-n63-k10 | * 1496 | 2033 | 1510 | 0.9 | 25.7 | 7145.6 | 4389.1 | 2532.6 | 1541 | 24.2 | 17.9 |
| B-n64-k9 | * 861 | 1370 | 864 | 0.4 | 36.9 | 924.0 | 578.5 | 322.2 | 877 | 36.0 | 28.4 |
| B-n66-k9 | * 1316 | 1668 | 1316 | 0.0 | 21.1 | 5573.9 | 1556.9 | 3934.3 | 1340 | 19.7 | 13.4 |
| B-n67-k10 | * 1032 | 1481 | 1037 | 0.5 | 30.0 | 1215.7 | 475.6 | 701.2 | 1040 | 29.8 | 18.5 |
| B-n68-k9 | * 1272 | 1558 | 1275 | 0.2 | 18.2 | 7108.8 | 4421.4 | 2540.1 | 1317 | 15.5 | 13.5 |
| B-n78-k10 | * 1221 | 1527 | 1260 | 3.2 | 17.5 | 4620.3 | 1895.1 | 2627.7 | 1287 | 15.7 | 11.2 |

**Table 2.** (cond.)

| | | Solution values | | | | Computing times | | | No recombination | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Problem | Best | Start | SERR | %err | %imp | Total | Cplex | Recomb. | Sol. | %imp | Time |
| E-n51-k5 | * 521 | 630 | 521 | 0.0 | 17.3 | 34.6 | 0.9 | 27.3 | 526 | 16.5 | 2.8 |
| E-n76-k7 | * 682 | 853 | 690 | 1.2 | 19.1 | 131.4 | 2.6 | 122.2 | 694 | 18.6 | 11.6 |
| E-n76-k8 | * 735 | 956 | 738 | 0.4 | 22.8 | 1302.8 | 75.1 | 1173.8 | 740 | 22.6 | 15.6 |
| E-n76-k10 | * 830 | 963 | 831 | 0.1 | 13.7 | 2941.4 | 630.6 | 2181.0 | 867 | 10.0 | 4.6 |
| E-n76-k14 | * 1021 | 1294 | 1032 | 1.1 | 20.2 | 5926.1 | 3795.2 | 2009.6 | 1064 | 17.8 | 14.8 |
| E-n101-k8 | * 815 | 969 | 830 | 1.8 | 14.3 | 1479.4 | 11.4 | 1442.0 | 835 | 13.8 | 10.6 |
| E-n101-k14 | 1071 | 1354 | 1101 | 2.8 | 18.7 | 4167.8 | 1452.4 | 2694.2 | 1099 | 18.8 | 10.0 |
| P-n50-k8 | * 631 | 867 | 643 | 1.9 | 25.8 | 2117.2 | 1690.5 | 326.2 | 650 | 25.0 | 22.8 |
| P-n55-k10 | * 694 | 889 | 698 | 0.6 | 21.5 | 944.8 | 261.0 | 583.2 | 703 | 20.9 | 15.9 |
| P-n60-k10 | * 744 | 955 | 744 | 0.0 | 22.1 | 807.6 | 165.8 | 570.9 | 764 | 20.0 | 8.4 |
| P-n60-k15 | * 968 | 1245 | 968 | 0.0 | 22.2 | 869.8 | 193.3 | 574.7 | 1008 | 19.0 | 4.8 |
| P-n65-k10 | * 792 | 922 | 800 | 1.0 | 13.2 | 531.8 | 41.2 | 444.3 | 809 | 12.3 | 6.9 |
| P-n70-k10 | * 827 | 1057 | 827 | 0.0 | 21.8 | 1658.4 | 471.2 | 1099.4 | 876 | 17.1 | 11.3 |
| P-n101-k4 | * 681 | 788 | 693 | 1.8 | 12.1 | 1374.2 | 0.1 | 1343.2 | 687 | 12.8 | 7.3 |
| F-n72-k4 | * 237 | 369 | 238 | 0.4 | 35.5 | 406.4 | 4.9 | 395.7 | 250 | 32.2 | 21.9 |
| E051-05e | * 524.61 | 633.38 | 524.61 | 0.0 | 17.2 | 32.9 | 1.3 | 25.2 | 527.67 | 16.7 | 6.7 |
| E076-10e | 835.26 | 965.86 | 848.55 | 1.6 | 12.1 | 2907.1 | 1040.0 | 1766.5 | 858.79 | 11.1 | 7.6 |
| E101-08e | 826.14 | 981.71 | 831.91 | 0.7 | 15.3 | 6153.5 | 47.5 | 5985.2 | 846.61 | 13.8 | 9.2 |
| E101-10c | 819.56 | 1142.59 | 819.56 | 0.0 | 28.3 | 3697.9 | 175.9 | 3460.6 | 880.45 | 22.9 | 15.4 |

- **Sol.** is the solution value found by the algorithm;
- **%imp** is the percentage improvement of the value found by the algorithm with respect to the initial solution value;
- **Time** is the total computing time required by the algorithm to obtain its final solution.

The tables show that our refining algorithm is able to improve substantially the initial solutions. The average improvement is about 3.5% and 24.4% for the initialization procedures FJ and SWEEP, respectively. With respect to the best known solutions available in the literature, algorithm SERR is able to find the best solution for 13 instances by starting with procedures FJ or SWEEP. With respect to the values given in [37] and corresponding to the best solutions found by heuristic and metaheuristic algorithms, algorithm SERR delivers an improved solution for instances A-n62-k8, P-n50-k8, P-n60-k10, P-n60-k15 (starting both from FJ and SWEEP), and P-n70-k10 (starting with SWEEP), the values reported in [37] for these instances being, respectively, 1290, 649, 756, 1033, 834.

Tables 3 and 4 address DCVRP instances, with a larger time-limit allowed; columns have the same meaning as in the previous tables. In the tables, (*) denotes instances for which the refining algorithm started from an infeasible initial solution. The performance of SERR for these instances is very satisfactory, in that the method is always able to recover the initial infeasibility (with a single exception arising when starting with the FJ solution) and to approach very closely the best-known solution. In one case, D101-11c, the method even improves the best-known solution from the literature.

Table 5 gives the solution values and times for the ACVRP instances; for these instances, only the FJ starting procedure is used—SWEEP heuristic requires customer coordinates, that are not specified in the asymmetric instances. Again, SERR proves quite effective in improving the initial (very poor) FJ solution, and delivers solution that are (on average) 3.7 % worse than the best-known (actually, provable optimal) ones.

For the second class of experiments, in which we start from an extremely-good feasible solution taken from the literature, the following CVRP and DCVRP instances have been considered:

- *real cost instances*: the 14 standard test problems proposed by Christofides and Eilon [6] and Christofides, Mingozzi and Toth [7];
- *rounded integer cost instances*: the same 14 standard test problems as before, plus instances E-n101-k14, M-n151-k12, M-n151-k12a, and D200-18c.

Problems M-n151-k12 and M-n151-k12a actually correspond to the same instance, the only difference being the order in which the customers are given on input. The corresponding initial solutions are taken from Gendreau, Hertz and Laporte [21] for the 14 standard instances, from Taillard's web page [41] for instances M-n151-k12 and D200-18c, and from Vigo's VRPLIB web page [44] for instances M-n151-k12a and E-n101-k14.

Table 6 reports the results obtained by algorithm SERR for the instances for which it is able to improve the corresponding initial solution. The columns have the same meaning as in the previous tables; a negative value for %err means that the new solution is strictly better than the previous best-known solution from the literature. For all the integer rounded cost DCVRP instances we assume that the starting solution, taken from

**Table 3.** Complete and no recombination method for DCVRP instances, with FJ initial solution. (†) infeasible initial solution

| | | Solution values | | | | Computing times | | | No recombination | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Problem | Best | Start | SERR | %err | %imp | Total | Cplex | Recomb. | Sol. | %imp | Time |
| D051-06c | 548 | †559 | 556 | 1.5 | – | 13.8 | <0.1 | 10.9 | 556 | – | 4.0 |
| D076-11c | 907 | †897 | 909 | 0.2 | – | 8887.5 | 5607.0 | 2253.7 | – | – | 62.2 |
| D101-09c | 856 | †875 | 863 | 0.8 | – | 1656.2 | 6.3 | 1625.3 | 861 | – | 70.6 |
| D101-11c | 866 | †858 | 865 | −0.1 | – | 11733.5 | 3861.1 | 7736.4 | 876 | – | 50.1 |
| D051-06c | 555.43 | †564.41 | 556.68 | 0.2 | – | 320.1 | 2.2 | 257.6 | 556.68 | – | 12.0 |
| D076-11c | 909.68 | †902.22 | – | – | – | 9352.9 | 7874.5 | 1395.6 | – | – | 145.6 |
| D101-09c | 865.94 | †885.90 | 871.93 | 0.7 | – | 13671.0 | 49.1 | 13396.2 | 892.79 | – | 58.4 |
| D101-11c | 866.37 | †859.49 | 867.65 | 0.1 | – | 21465.6 | 9022.1 | 12212.1 | 907.85 | – | 94.1 |

**Table 4.** Complete and no recombination method for DCVRP instances, with SWEEP initial solution. (†) infeasible initial solution

| | | Solution values | | | | Computing times | | | No recombination | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Problem | Best | Start | SERR | %err | %imp | Total | Cplex | Recomb. | Sol. | %imp | Time |
| D051-06c | 548 | †630 | 559 | 2.0 | – | 74.3 | 12.3 | 48.2 | 568 | – | 4.5 |
| D076-11c | 907 | †963 | 918 | 1.2 | – | 6385.7 | 4155.1 | 2109.9 | – | – | 90.1 |
| D101-09c | 856 | †969 | 869 | 1.5 | – | 1405.9 | 106.9 | 1277.2 | 916 | – | 129.0 |
| D101-11c | 866 | †1143 | 865 | −0.1 | – | 28588.2 | 12550.8 | 15788.8 | 928 | – | 113.8 |
| D051-06c | 555.43 | †633.38 | 560.24 | 0.9 | – | 264.8 | 24.1 | 195.5 | 570.81 | – | 12.5 |
| D076-11c | 909.68 | †965.86 | 911.76 | 0.2 | – | 15838.5 | 9787.2 | 5713.2 | – | – | 295.7 |
| D101-09c | 865.94 | †981.71 | 874.96 | 1.0 | – | 9175.6 | 169.7 | 8843.9 | 880.10 | – | 191.1 |
| D101-11c | 866.37 | †1142.59 | 878.52 | 1.4 | – | 7579.6 | 4172.5 | 3337.1 | 879.32 | – | 196.2 |

**Table 5.** Complete and no recombination method for ACVRP instances, with FJ initial solution. (*) provable optimal solution value

| | | Solution values | | | | Computing times | | | No recombination | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Problem | Best | Start | SERR | %err | %imp | Total | Cplex | Recomb. | Sol. | %imp | Time |
| A034-02f | * 1406 | 1607 | 1409 | 0.2 | 12.3 | 6.7 | 0.1 | 0.1 | 1530 | 4.8 | 0.1 |
| A036-03f | * 1644 | 1981 | 1724 | 4.9 | 13.0 | 1.5 | <0.1 | <0.1 | 1732 | 12.6 | 0.1 |
| A039-03f | * 1654 | 2100 | 1780 | 7.6 | 15.2 | 21.1 | 0.4 | 0.4 | 1809 | 13.9 | 2.4 |
| A045-03f | * 1740 | 2150 | 1829 | 5.1 | 14.9 | 16.2 | 0.2 | 2.0 | 1829 | 14.9 | 2.2 |
| A048-03f | * 1891 | 2262 | 1962 | 3.8 | 13.3 | 4.7 | <0.1 | 1.6 | 1997 | 11.7 | 2.6 |
| A056-03f | * 1739 | 2071 | 1777 | 2.2 | 14.2 | 8.0 | <0.1 | 5.5 | 1806 | 12.8 | 0.7 |
| A065-03f | * 1974 | 2534 | 2037 | 3.2 | 19.6 | 160.9 | 0.5 | 112.2 | 2126 | 16.1 | 3.4 |
| A071-03f | * 2054 | 2406 | 2111 | 2.8 | 12.3 | 113.5 | <0.1 | 96.1 | 2160 | 10.2 | 13.3 |

**Table 6.** Complete method: improvements from good CVRP/DCVRP solutions from the literature. (*) provable optimal solution value. (#) solution with one empty route

| | | Solution values | | | | | |
|---|---|---|---|---|---|---|---|
| Problem | Best | Start | Source | SERR | %err | %imp | Comp. time |
| E076-10e | * 830 | 832 | [22] | 831 | 0.1 | 0.1 | 279.4 |
| E101-10c | * 820 | 824 | [22] | 820 | 0.0 | 0.5 | 18.3 |
| E121-07c | * 1034 | 1035 | [22] | 1034 | 0.0 | 0.1 | 88.7 |
| E151-12c | 1016 | 1024 | [22] | 1022 | 0.6 | 0.2 | 461.2 |
| E200-17c | 1316 | 1316 | [22] | 1307 | −0.7 | 0.7 | 48487.7 |
| D076-11c | 907 | 907 | [22] | 905 | −0.2 | 0.2 | 178.1 |
| D101-11c | 866 | 866 | [22] | 865 | −0.1 | 0.1 | 1273.5 |
| D121-11c | 1529 | 1529 | [22] | 1526 | −0.2 | 0.2 | 26622.1 |
| D151-14c | 1180 | 1180 | [22] | 1161 | −1.6 | 1.6 | 44578.4 |
| D200-18c | 1404 | 1404 | [22] | 1398 | −0.4 | 0.4 | 4074.8 |
| E076-10e | 835.26 | 836.37 | [22] | 835.26 | 0.0 | 0.1 | 380.7 |
| E101-10c | 819.56 | 822.85 | [22] | 819.56 | 0.0 | 0.4 | 20.0 |
| E121-07c | 1042.11 | 1043.94 | [22] | 1043.42 | 0.1 | <0.1 | 114.8 |
| E151-12c | 1028.42 | 1034.90 | [22] | 1034.50 | 0.6 | <0.1 | 396.5 |
| E200-17b | 1291.29 | 1311.35 | [44] | 1305.35 | 1.1 | 0.5 | 18386.0 |
| D121-11c | 1541.14 | 1551.63 | [22] | 1546.10 | 0.3 | 0.4 | 232465.9 |
| D151-14c | 1162.55 | 1189.79 | [22] | 1178.02 | 1.3 | 1.0 | 7431.4 |
| D200-18c | 1395.85 | 1421.88 | [22] | 1416.47 | 1.5 | 0.4 | 42261.9 |
| E-n101-k14 | 1071 | 1076 | [44] | 1067 | -0.4 | 0.8 | 2865.7 |
| M-n151-k12 | 1016 | 1016 | [41] | 1015 | -0.1 | 0.1 | 377.4 |
| M-n151-k12a | 1016 | 1023 | [44] | 1022 | 0.6 | 0.1 | 11090.2 |
| D200-18c | 1395.85 | 1395.85 | [41] | 1352.01 | -3.2 | 3.1 | 79820.4 |
| D200-18c | 1395.85 | 1395.85 | [41] | # 1347.61 | -3.6 | 3.5 | 113447.2 |

[21], corresponds to the best known one—unfortunately, for the integer-rounded-cost DCVRP instances we could not find an explicit statement of this property in the literature. The table shows that SERR delivers an improved solution in 22 out of the 32 cases, while obtains a new best-known solution for 9 hard instances from the literature. Moreover, a modified SERR implementation allowing for empty routes delivers a new best-known solution for problem D200-18c. Finally, for both M-n151-k12 and M-n151-k12a algorithm SERR is able to improve the previous-best heuristic solution–the best solution value for both being 1015.

According to the previous table, the SERR method is very effective in improving the starting solution, even if it is of very-good quality. The computational effort appears quite substantial in some cases. However, as shown in Figures 5 and 6, the quality of the incumbent SERR solution improves quickly at the very beginning of the computation, so one could think of imposing a much shorter time limit without a significant deterioration of the final solution. Moreover, our computational experiments show that the no-recombination method often has an acceptable performance, so a faster SERR implementation could be obtained by applying the fast no-recombination method first, so as to quickly improve the incumbent solution, and only afterwards the more time-consuming complete method.

We finally addressed larger instances with more than 200 customers, in the attempt of improving the corresponding best-known solution values reported in the literature. In order to reduce the computational time required by our method, in the node recombination phase described in Subsection 4.4 we inhibited the generation of derived sequences of

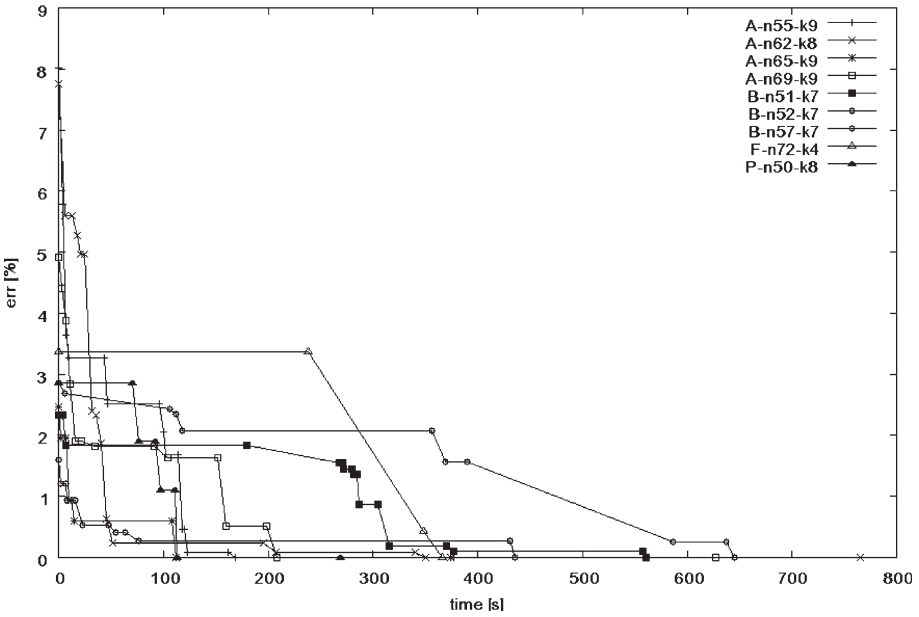


**Fig. 5.** Time evolution of the SERR solution for various CVRP instances, with FJ initial solution

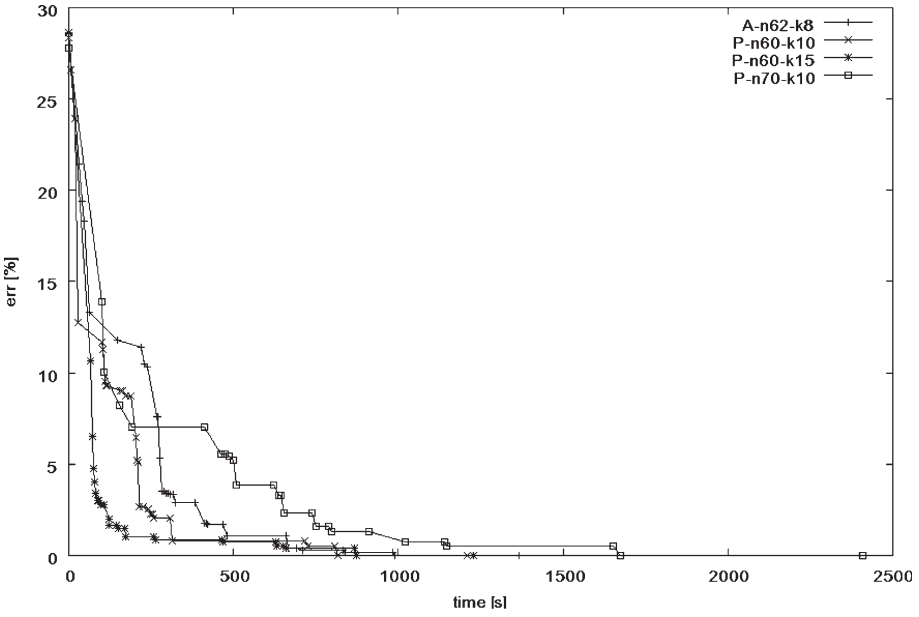


**Fig. 6.** Time evolution of the SERR solution for various CVRP instances, with SWEEP initial solution

length larger than 2, and used the following alternative parameter setting: $RC_{max} = L_{max} = 3$, $N_{min} = 2$, $N_{max} = 5$, and $\delta = 2$. The following 13 instances have been considered: E241-22k, E253-27k, E256-14k, E301-28k, E321-30k, E324-16k,

E361-33k, E397-34k, E400-18k, E421-41k, E481-38k, and E484-19k from [24], and Tai385 from [41]. The SERR method was able to improve 4 solutions, namely, E324-16k (from the previous best-known value 742.03 to 741.70, in 61661.9 seconds), E361-33k (from 1366.8579 to 1366,8578, in 1032.0 seconds), E400-18k (from 918.4464542 to 918.4151609, in 5584.76 seconds), and Tai385 (from 24431.44 to 24422.50, in 152286.5 seconds).[2] The new best-known solutions are given in the appendix.

## 6. Conclusions and future research directions

We have proposed a new refinement heuristic for DCVRP. The method is an elaboration of a refinement procedure originally proposed by Sarvanov and Doroshko [40] (SD) for the pure Travelling Salesman Problem (TSP).

Our approach goes far beyond the original SD scheme, and is based on a more sophisticated node removal policy followed by a procedure to construct a large number of new potential sequences through the extracted nodes.

Computational results on a large set of capacitated VRP instances from the literature (with symmetric/asymmetric costs and with/without distance constraints) show that the SERR method is quite effective. In 13 cases, some of which are reported in the Appendix, it was even able to improve the best-known solution reported in the literature.

Future directions of work should address the possibility of extending the SERR method to even more constrained VRP versions. An obvious extension is to consider route-dependent limits on the capacity and/or on the total-distance travelled. arc costs. Also interesting is the adaptation of the SERR method to scheduling problems (including the crew and vehicle scheduling problems addressed, e.g., in [15], for which the method can be viewed as a generalization of the refinement heuristic proposed in [9]), or to the VRP with backhauls [43]. An even more intriguing extension is to consider the DCVRP with *precedence constraints*, and important variant where a set of precedences among the nodes is specified. In the SERR context, these additional constraints would lead to (a) removing some variables $x_{si}$, in case the allocation of sequence $s$ into insertion point $i$ is impossible due to some precedences between a node covered in sequence $s$ and one of the nodes in the restricted route that are visited before the insertion point $i$, and (b) introducing new constraints among the $x_{si}$ variables corresponding to the insertion points $i$ belonging to a same route, stipulating incompatibility conditions of the type $x_{s'i'} + x_{si} \leq 1$ whenever there exists a precedence between a node in $s'$ and a node $s$ which is incompatible with the relative position of the insertion points $i$ and $i'$ in the route (these constraints possibly need to be strengthened by lifting techniques).

*Acknowledgements.* We thank David Mester for having provided us with the best-known solutions for the instances given in [24]. Thanks are also due to the two anonymous referees for their helpful comments.

## References

1. Ahuja, R.K., Ergun, O., Orlin, J.B., Punnen, A.B.: A survey of very large-scale neighborhood search techniques. Discrete Applied Math. **123**, 75–102 (2002)

[2] With a slightly different tuning, for problem E324-16k we found a solution of value 742.02 within 45.9 seconds, whereas for problem Tai385 we found a solution of value 24427.51 within 35842.1 seconds.

2. Balas, E., Simonetti, N.: Linear time dynamic programming algorithms for new classes of Restricted TSPs. Informs J. Comput. **13**, 56–75 (2001)
3. Baldacci, R., Hadjiconstantinou, E.A., Mingozzi, A.: An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. Oper. Res. 2004 (forthcoming)
4. Burkard, R.E., Deineko, V.G.: Polynomially solvable cases of the traveling salesman problem and a new exponential neighborhood. Comput. **54**, 191–211 (1995)
5. Cordeau, J.-F., Gendreau, M., Hertz, A., Laporte, G., Sormany, J.-S.: New heuristics for the vehicle routing problem. In: A. Langevin, D. Riopel (eds.). Logistics Systems: Design and Optimization, Kluwer, Boston, 2005 forthcoming
6. Christofides, N., Eilon, S.: Expected distances in distribution problems. Oper. Res. Quarterly **20** (4), 437–443 (1969)
7. Christofides, N., Mingozzi, A., Toth, P.: The vehicle routing problem. Combinatorial Optimization (Wiley, Chichester), 1979
8. Deineko, V.G., Woeginger, G.J.: A study of exponential neighborhoods for the travelling salesman problem and for the quadratic assignment problem. Math. Programming **87**, 519–542 (2000)
9. Dell'Amico, M., Fischetti, M., Toth, P.: Heuristic algorithms for the multiple depot vehicle scheduling problem. Management Sci. **39**, 115–125 (1993)
10. Ergun, O.: New neighborhood search algorithms based on exponentially large neighborhoods. Operations Research Center Thesis, MIT, 2001
11. Ergun, O., Orlin, J.B.: Two dynamic programming methodologies in very large scale neighborhood search applied to the Traveling Salesman Problem. Working Paper 4463-03, MIT Sloan School of Management, 2003
12. Ergun, O., Orlin, J.B., Steele-Feldman, A.: A computational study on a large-scale neighborhood search algorithm for the vehicle routing problem with capacity and distance constraints. Working paper, School of Ind. and Sys. Engr. Georgia Institute of Technology, Atlanta, 2002
13. Ergun, O., Orlin, J.B., Steele-Feldman, A.: Creating very large scale neighborhoods out of smaller ones by compounding moves: a study on the Vehicle Routing Problem. Working Paper 4393-02, MIT Sloan School of Management, 2002
14. Firla, R.T., Spille, B., Weismantel, R.: Exponentialirreducible neighborhoods for combinatorial optimization problems. Math. Meth. Oper. Res. **56**, 29–44 (2002)
15. Fischetti, M., Lodi, A., Martello, S., Toth, P.: A polyhedral approach to simplified crew scheduling and vehicle scheduling problems. Management Sci. **47** (6), 833–850 (2001)
16. Fischetti, M., Toth, P., Vigo, D.: A branch and bound algorithm for the capacitated vehicle routing problem on directed graphs. Oper. Res. **42**, 846–859 (1994)
17. Fisher, M.: Vehicle routing. In: M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser (eds.). Network Routing, Handbooks in OR & MS Vol. **8**, Elsevier, Amsterdam, 1995
18. Fisher, M.L., Jaikumar, R.: A generalized assignment heuristic for vehicle routing. Networks **11**, 109–124 (1981)
19. Fukasawa, R., Poggi de Aragão, M., Reis, M., Uchoa, E: Robust branch-and-cut-and-price for the capacitated vehicle routing problem, Relatórios de Pesquisa em Engenharia de Produção, RPEP **3** (8), Universidade Federal Fluminense, 2003
20. Gillett, B.E., Miller, L.R.: A heuristic algorithm for the vehicle dispatch problem. Oper. Res. **22**, 340–349 (1974)
21. Gendreau, M., Hertz, A., Laporte, G.: A Tabu Search Heuristic for the VRP, Technical Reoprt CRT-777, 1991
22. Gendreau, M., Hertz, A., Laporte, G.: A tabu search heuristic for the vehicle routing problem. Management Sci. **40**, 1276–1290 (1994)
23. Gendreau, M., Laporte, G., Potvin, J.-Y.: Metaheuristics for the capacitated VRP. In: P. Toth, D. Vigo (eds.). The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications, 2002
24. Golden, B.L., Wasil, E.A., Kelly, J.P., Chao, I-M.: Metaheuristics in Vehicle Routing. In: T.G. Crainic, G. Laporte (eds.). Fleet Management and Logictics. Kluwer, Boston, 1998 pp. 33–56
25. Glover, F., Punnen, A.P.: The traveling salesman problem: New solvable cases and linkages with the development of approximation algorithms. J. Oper. Res. Soc. **48**, 502–510 (1997)
26. Gutin, G.M.: On an approach to solving the traveling salesman problem, Proc. The USSR Conference on System Research (Moscow, USSR), 1984 pp. 184–185 (in Russian)
27. Gutin, G.M.: Exponential neighborhood local search for the traveling salesman problem. Comput. Oper. Res. **26**, 313–320 (1999)
28. Gutin, G., Yeo, A., Zverovitch, A.: Exponential neighborhoods and domination analysis for the TSP. In: G. Gutin, A. Punnen (eds.). The Traveling Salesman Problem and its Variations. Kluwer, 2002 pp. 223–256

29. ILOG Cplex 8.1: user's manual and reference manual, ILOG, S.A., http://www.ilog.com/, 2003
30. ILOG Concert Technology 1.2: User's manual and reference manual, ILOG, S.A., http://www.ilog.com/, 2003
31. Johnson, D.S., McGeoch, L.A.: Experimental analysis of heuristics for the STSP. In: G. Gutin, A.P. Punnen (eds.). The Traveling Salesman Problem and Its Variations Kluwer Academic Publishers, 9, pp. 369–487, 2002
32. Laporte, G., Semet, F.: Classical heuristics for the capacitated VRP. In: P. Toth, D. Vigo (eds.). The Vehicle Routing Problem SIAM Monographs on Discrete Mathematics and Applications, Chap. 5, 2002
33. Li, F., Golden, B.L., Wasil, E.A.: Very large-scale vehicle routing: New test problems, algorithms, and results. Comput. Oper. Res. 2004 (forthcoming)
34. Lysgaard, J., Letchford, A.N., Eglese, R.W.: A new branch-and-cut algorithm for the capacitated vehicle routing problem. Math. Prog. **100** (2), 423–442 2004
35. Mester, D., Braysy, O.: Active guided evolution strategies for the large scale vehicle routing problems with time windows. Comput. Oper. Res. 2004 (forthcoming)
36. Punnen, A.P.: The Traveling Salesman Problem: new polynomial approximation and domination analysis. J. Information Optim. Sci. **22**, 191–206 (2001)
37. Ralphs, T.: Branch and Cut and Price Applications: Vehicle Routing, http://branchandcut.org/VRP/
38. Rego, C., Glover, F.: Local search and metaheuristcs. In: G. Gutin, A. Punnen (eds.). The Traveling Salesman Problem and its Variations, Kluwer, 2002 pp. 309–368
39. Reimann, M., Doerner, K., Hartl, R.F.: D-Ants: Savings based ants divide and conquer the vehicle routing problem. Comput. Oper. Res. 2004 (forthcoming)
40. Sarvanov, V.I., Doroshko, N.N.: The approximate solution of the travelling salesman problem by a local algorithm with scanning neighborhoods of factorial cardinality in cubic time, (in Russian), Software: Algorithms and Programs 31, Mathematical Institute of the Belorussian Academy of Sciences, Minsk, 1981, pp. 11–13
41. Taillard, E.: Eric Taillard's Page, Vehicle Routing Instances, http://ina.eivd.ch/collaborateurs/etd/problemes.dir/vrp.dir/vrp.html.
42. Tarantilis, C.-D., Kiranoudis, C.T.: Bone Route: An adaptive memory-based method for effective fleet management. Ann. Oper. Res. **115**, 227–241 (2002)
43. Toth, P., Vigo, D.: An overview of vehicle routing problems. In: P. Toth, D. Vigo (eds.). The Vehicle Routing Problem, SIAM Monographs on Discrete Mathematics and Applications, 2002
44. Vigo, D.: VRPLIB: A Vehicle Routing Problem LIBrary, http://www.or.deis.unibo.it/research_pages/ORinstances/VRPLIB/VRPLIB. html
45. Wenger, K.M.: Generic cut generation methods for routing problems. Ph.D Dissertation, Institute of Computer Science, University of Heidelberg, 2003

## 7. Appendix: Improved solutions

We next present some improved solutions found by our SERR method.

**New best-known solution for instance E-n101-k14, having cost 1067**

```
R1: 61, 16, 86, 38, 44, 91, 98
R2: 1, 51, 9, 81, 33, 79, 50
R3: 58, 2, 57, 42, 14, 43, 15, 41, 22, 74, 73
R4: 69, 70, 30, 32, 90, 63, 10, 31
R5: 80, 24, 29, 78, 34, 35, 71, 65, 66, 20
R6: 12, 68, 3, 77, 76, 28
R7: 82, 48, 47, 19, 7, 52
R8: 88, 62, 11, 64, 49, 36, 46, 8, 18
R9: 96, 99, 93, 85, 100, 37, 92
R10: 54, 55, 25, 39, 67, 23
R11: 6, 59, 5, 84, 17, 45, 83, 60, 89
R12: 40, 21, 72, 75, 56, 4, 26
R13: 94, 95, 97, 87, 13
R14: 53, 27
```

**New best-known solution for instance M-n151-k12, having cost 1015**

```
R1: 69, 101, 70, 30, 20, 128, 66, 71, 65, 136, 35, 135, 34, 78, 129, 79
R2: 137, 2, 115, 145, 41, 22, 133, 23, 56, 75, 74, 72, 73, 21, 40
R3: 138, 109, 54, 130, 55, 25, 67, 39, 139, 4, 110, 149, 26
R4: 132, 1, 122, 51, 103, 9, 120, 81, 33, 102, 50, 111
R5: 27, 127, 31, 10, 108, 131, 32, 90, 63, 126, 62, 148, 88
R6: 18, 114, 46, 124, 47, 36, 143, 49, 64, 11, 107, 19, 123, 7, 146
R7: 89, 118, 60, 83, 125, 45, 8, 82, 48, 106, 52
R8: 147, 5, 84, 17, 113, 86, 140, 38, 43, 15, 57, 144, 58
R9: 112, 53, 105
R10: 28, 76, 116, 77, 3, 121, 29, 24, 134, 80, 150, 68, 12
R11: 6, 96, 104, 99, 61, 16, 141, 44, 119, 14, 142, 42, 87, 97, 117
R12: 94, 59, 93, 85, 91, 100, 37, 98, 92, 95, 13
```

**New best known solution for instance D200-18c, having cost 1352.01**

```
R1: 110, 155, 4, 139, 187, 39, 186, 56, 197, 198, 180, 53
R2: 40, 73, 171, 74, 133, 22, 41, 145, 115, 178, 2, 58, 152
R3: 12, 109, 177, 80, 150, 163, 24, 134, 54, 195, 26, 105
R4: 27, 167, 127, 190, 88, 182, 194, 106, 52, 146
R5: 149, 179, 130, 165, 55, 25, 170, 67, 23, 75, 72, 21
R6: 156
R7: 37, 100, 192, 119, 14, 38, 140, 44, 141, 191, 91, 193, 59
R8: 6, 96, 104, 99, 93, 85, 98, 151, 92, 95, 94, 183
R9: 60, 118, 84, 17, 113, 86, 16, 61, 173, 5, 147
R10: 137, 144, 57, 15, 43, 142, 42, 172, 87, 97, 117, 13, 112
R11: 7, 123, 19, 49, 143, 36, 47, 168, 124, 48, 82, 153
R12: 89, 166, 83, 199, 125, 45, 46, 174, 8, 114, 18
R13: 154, 138, 184, 116, 196, 76, 28
R14: 68, 121, 29, 169, 34, 164, 78, 129, 79, 158, 3, 77
R15: 111, 50, 102, 157, 33, 9, 103, 66, 188, 20, 122, 1
R16: 31, 159, 126, 63, 181, 64, 11, 175, 107, 62, 148
R17: 132, 69, 101, 70, 30, 128, 160, 131, 32, 90, 108, 189, 10, 162
R18: 185, 81, 120, 135, 35, 136, 65, 71, 161, 51, 176
```

**New best known solution for instance D200-18c, having cost 1347.61 (with one empty route)**

```
R1: 110, 155, 4, 139, 187, 39, 186, 56, 197, 198, 180, 53
R2: 40, 73, 171, 74, 133, 22, 41, 145, 115, 178, 2, 58, 152
R3: 12, 109, 177, 150, 80, 163, 24, 134, 54, 195, 26, 105
R4: 149, 179, 130, 165, 55, 25, 170, 67, 23, 75, 72, 21
R5: 27, 167, 127, 190, 88, 182, 194, 106, 52, 146
R6: 37, 100, 192, 119, 14, 38, 140, 44, 141, 191, 91, 193, 59
R7: 183, 94, 95, 92, 151, 98, 85, 93, 104, 99, 96, 6
R8: 60, 118, 84, 17, 113, 86, 16, 61, 173, 5, 147
R9: 137, 144, 57, 15, 43, 142, 42, 172, 87, 97, 117, 13, 112, 156
R10: 153, 82, 48, 124, 168, 47, 36, 143, 49, 19, 123, 7
R11: 89, 166, 83, 199, 125, 45, 46, 174, 8, 114, 18
R12: 138, 154, 184, 116, 196, 76, 28
R13: 77, 3, 158, 79, 129, 78, 164, 34, 169, 29, 121, 68
R14: 111, 50, 102, 157, 33, 9, 103, 66, 188, 20, 122, 1
R15: 31, 159, 126, 63, 181, 64, 11, 175, 107, 62, 148
R16: 132, 69, 101, 70, 30, 128, 160, 131, 32, 90, 108, 10, 189, 162
R17: 185, 81, 120, 135, 35, 136, 65, 71, 161, 51, 176
```

**New best known solution for instance E324-16k, having cost 741.70**

```
R1: 129, 163, 181, 213, 228, 241, 254, 276, 285, 293, 300, 306, 311,
315, 318, 314, 309, 303, 297, 290, 281, 271, 261, 237, 224, 209, 194,
178, 143, 83
R2: 61, 85, 113, 145, 180, 197, 212, 227, 240, 253, 265, 275, 284, 274,
264, 252, 239, 226, 196, 162, 128, 98, 72
R3: 164, 182, 199, 215, 230, 243, 267, 278, 287, 295, 302, 308, 313,
317, 320, 322, 323, 321, 319, 316, 312, 307, 301, 294, 286, 277, 266,
255, 242, 229, 214, 198, 146, 99
R4: 127, 161, 195, 210, 238, 250, 262, 272, 282, 291, 298, 304, 310,
305, 299, 292, 283, 273, 263, 251, 225, 211, 179, 144, 112, 84
R5: 8, 9, 5, 2
R6: 4, 1
R7: 17, 23, 30, 39, 49, 60, 50, 40, 31, 24
R8: 13, 19, 14, 20, 26, 33, 25, 18, 12
R9: 32, 41, 51, 73, 86, 114, 130, 147, 165, 183, 200, 184, 166, 148,
131, 115, 100, 74, 62, 42
R10: 71, 97, 111, 126, 160, 177, 193, 208, 223, 236, 249, 260, 248, 235,
222, 207, 192, 176, 159, 142, 110, 96, 70, 59
R11: 95, 109, 124, 140, 157, 174, 190, 206, 220, 233, 246, 258, 269,
279, 288, 296, 289, 280, 270, 259, 247, 234, 221, 191, 175, 158, 141,
125, 82, 48
R12: 38, 47, 57, 68, 80, 93, 107, 122, 138, 155, 172, 188, 204, 219,
205, 189, 173, 156, 139, 123, 108, 94, 81, 69, 58
R13: 3, 6, 10, 15, 22, 16, 11, 7
R14: 52, 87, 101, 116, 132, 149, 167, 185, 202, 217, 201, 216, 231, 244,
256, 268, 257, 245, 232, 218, 203, 186, 168, 150, 133, 117, 102, 88,
75, 63
R15: 27, 35, 44, 54, 65, 77, 90, 104, 119, 135, 152, 170, 187, 169, 151,
134, 118, 103, 89, 76, 64, 53, 43, 34
R16: 21, 28, 36, 45, 55, 66, 78, 91, 105, 120, 136, 153, 171, 154, 137,
121, 106, 92, 79, 67, 56, 46, 37, 29
```

**New best known solution for instance E361-33k, having cost 1366.8578**

```
R1: 14, 23, 24, 25, 84, 85, 67, 80, 7, 20, 19
R2: 34, 35, 36, 9, 101, 69, 160, 100, 40, 39
R3: 41, 161, 221, 189, 276, 335, 334, 275, 216, 129, 156, 96
R4: 140, 200, 260, 247, 325, 307, 320, 319, 318, 259, 199
R5: 172, 232, 292, 251, 357, 311, 352, 351, 350, 291, 231
R6: 212, 213, 272, 331, 332, 273, 188, 208, 128, 153, 93
R7: 3, 63, 123, 142, 91, 82, 31, 22
R8: 207, 267, 268, 248, 333, 308, 328, 327, 326, 321, 266
R9: 287, 338, 347, 348, 349, 310, 344, 250, 289, 288, 228
R10: 43, 44, 10, 104, 70, 164, 163, 103, 102, 42, 37
R11: 109, 169, 130, 229, 190, 284, 343, 342, 283, 224, 223
R12: 1, 61, 121, 174, 75, 16, 15, 114, 54
R13: 79, 139, 198, 253, 258, 313, 301, 241, 294, 195, 136
R14: 51, 52, 11, 112, 71, 117, 116, 57, 56, 55, 46
R15: 38, 47, 98, 107, 158, 125, 65, 5
R16: 220, 280, 281, 249, 336, 309, 341, 340, 339, 330, 279
R17: 255, 354, 315, 316, 317, 312, 360, 252, 257, 256, 196
R18: 45, 50, 105, 110, 165, 170, 225, 185, 218, 167, 108, 49, 48
R19: 21, 26, 27, 28, 87, 147, 148, 68, 88, 8, 33
R20: 60, 120, 72, 180, 239, 298, 353, 306, 346, 295, 286, 235, 175
R21: 17, 12, 76, 135, 234, 181, 193, 138, 133, 78, 73, 18, 13
R22: 32, 151, 202, 183, 243, 269, 214, 209, 154, 149, 94, 89, 29
R23: 53, 58, 113, 118, 173, 126, 66, 6
```

```
R24: 4, 64, 124, 184, 210, 150, 90, 30
R25: 144, 204, 264, 323, 324, 265, 187, 205, 127, 145
R26: 177, 131, 237, 191, 297, 356, 355, 296, 236, 176
R27: 168, 227, 278, 245, 305, 345, 290, 285, 230, 171, 111
R28: 77, 137, 132, 197, 192, 300, 359, 358, 299, 240, 179
R29: 74, 83, 134, 143, 194, 203, 254, 263, 314, 302, 242, 261, 206, 146
R30: 92, 152, 211, 262, 271, 322, 303, 329, 274, 215, 155, 95
R31: 106, 115, 166, 226, 186, 246, 293, 238, 233, 178, 119, 59
R32: 99, 159, 219, 270, 244, 304, 337, 282, 277, 222, 217, 162, 157, 97
R33: 81, 86, 141, 201, 182, 122, 62, 2
```

**New best known solution for instance Tai385, having cost 24422.50**

```
R1: 341, 342, 349, 346, 347, 348, 383, 375
R2: 336, 333, 334
R3: 246, 242, 228, 227
R4: 262, 261, 260, 259, 266, 276
R5: 139, 180, 179, 178, 190, 183, 182, 186, 185, 345, 378
R6: 193, 203, 205, 206, 213, 214, 216, 373, 217, 219, 32, 33, 34, 35, 5,
160, 161, 162, 163, 164
R7: 382, 358, 364
R8: 360, 357, 325
R9: 319, 281, 385, 49, 50, 57, 52
R10: 209, 210, 211, 212, 215, 365, 218, 220, 37, 39, 38, 40, 45, 384,
43, 42, 44, 48, 47, 46, 41, 36, 368, 192
R11: 300, 327, 359, 361, 362, 363, 323, 324
R12: 239, 238, 235, 233, 249, 250, 379
R13: 241, 236, 229, 230, 3, 223, 224, 231, 232, 234, 237, 240
R14: 314
R15: 351, 352, 221, 222
R16: 320, 279, 265, 254, 255, 380, 256, 251, 252, 248, 243, 244, 245,
247, 253, 374, 1
R17: 353, 354, 356, 355, 350, 225, 226
R18: 269, 338, 339, 340, 343, 208, 367, 202, 344, 189, 184, 181
R19: 308, 307
R20: 145, 121, 112, 113, 118, 115, 114, 146, 149, 151
R21: 142, 171, 381, 167, 165, 191, 199, 194, 176, 177, 175, 174, 173
R22: 302, 311
R23: 72, 69, 64, 65, 18, 19, 63, 62, 73, 81
R24: 88, 82, 80, 79, 78, 58, 59, 60, 61, 76, 77, 75, 74, 103, 90
R25: 275, 288, 294
R26: 310
R27: 136, 2, 140, 134, 133, 127, 128
R28: 305, 290, 289, 85, 54, 53, 283, 284, 291, 292, 321, 303, 309
R29: 96, 101, 102, 104, 105, 108, 130, 126, 97
R30: 188, 376, 124, 106, 71, 70, 107, 125, 98
R31: 304, 86, 84, 83, 56, 55, 187, 51, 282, 280, 257, 258, 286, 295
R32: 287, 263, 264, 277, 278, 285, 293
R33: 317
R34: 322, 301
R35: 274, 270, 332, 268
R36: 330, 331, 335
R37: 297, 267, 329
R38: 296, 273, 272, 298
R39: 138, 318
R40: 306, 87, 89, 100, 99, 312
R41: 94, 137, 132, 95, 93, 92, 91
R42: 315, 313
R43: 371, 110, 66, 68, 15, 17, 16, 29, 13, 24, 22, 23, 20, 21, 30, 31,
```

```
12, 11, 10, 9, 8, 7, 25, 26, 14, 370, 377, 67, 109, 122, 123, 129
R44: 135, 144, 152, 150, 155, 156, 157, 4, 158, 159, 366, 198, 201, 197,
200, 207, 204, 195, 196, 169, 172, 170
R45: 111, 119, 120, 28, 27, 6, 369, 117, 116, 147, 148, 154, 166, 153,
168, 143, 141, 131
R46: 299, 316, 372
R47: 326, 328, 337, 271
```

**New best known solution for instance E400-18k, having cost 918.4151609**

```
R1: 5, 9, 14, 20, 27, 43, 34, 26, 19, 13
R2: 53, 64, 76, 89, 103, 118, 134, 150, 168, 151, 169, 188, 208, 227,
246, 228, 209, 189, 170, 152, 135, 119, 104, 90, 77, 65, 54, 44, 35
R3: 33, 42, 51, 62, 61, 73, 86, 99, 85, 72, 60, 50, 41, 25
R4: 2, 8, 4
R5: 162, 200, 238, 288, 316, 328, 339, 349, 359, 368, 375, 381, 387,
392, 395, 397, 399, 398, 396, 393, 388, 382, 376, 369, 360, 350, 340,
329, 317, 303, 273, 257, 201, 163
R6: 1
R7: 114, 146, 182, 221, 240, 258, 274, 289, 304, 290, 305, 318, 330,
341, 351, 361, 352, 342, 331, 319, 306, 292, 276, 291, 275, 259, 241,
222, 202, 164, 130, 100
R8: 52, 63, 75, 88, 102, 117, 133, 149, 167, 186, 206, 187, 207, 226,
245, 263, 279, 262, 244, 225, 205, 185, 166, 148, 132, 116, 101, 87, 74
R9: 3, 6, 10, 15, 21, 29, 22, 16, 7
R10: 12, 18, 24, 32, 40, 31, 17, 11
R11: 113, 129, 145, 181, 220, 239, 256, 272, 287, 302, 315, 327, 338,
326, 314, 301, 286, 271, 255, 237, 219, 180, 144, 128, 98
R12: 161, 199, 218, 254, 270, 300, 313, 325, 337, 348, 358, 367, 374,
380, 386, 391, 394, 390, 385, 379, 373, 366, 357, 347, 336, 324, 312,
299, 285, 269, 253, 236, 198, 179
R13: 115, 131, 165, 184, 204, 224, 243, 261, 278, 294, 308, 321, 333,
344, 354, 363, 371, 378, 384, 389, 383, 377, 370, 362, 353, 343, 332,
320, 307, 293, 277, 260, 242, 223, 203, 183, 147
R14: 69, 95, 109, 124, 140, 157, 176, 195, 214, 232, 249, 265, 280, 295,
309, 296, 281, 266, 282, 267, 250, 233, 215, 196, 177, 158, 141, 125,
110, 70
R15: 23, 39, 49, 59, 71, 83, 97, 112, 127, 111, 96, 82, 58, 48, 30
R16: 38, 81, 94, 108, 123, 139, 156, 175, 194, 213, 231, 248, 264, 247,
229, 210, 190, 171, 153, 136, 120, 105, 91, 78, 66, 55, 45, 36, 28
R17: 47, 57, 68, 80, 93, 107, 122, 138, 155, 174, 193, 173, 192, 212,
230, 211, 191, 172, 154, 137, 121, 106, 92, 79, 67, 56, 46, 37
R18: 84, 43, 160, 178, 217, 235, 252, 284, 298, 311, 323, 335, 346, 356,
365, 372, 364, 355, 345, 334, 322, 310, 297, 283, 268, 251, 234, 216,
197, 159, 142, 126
```