Computers & Operations Research 40 (2013) 3197–3205

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Statistical analysis of distance-based path relinking for the capacitated vehicle routing problem

Kenneth Sörensen *, Patrick Schittekat

University of Antwerp, Faculty of Applied Economics, ANT/OR University of Antwerp Operations Research Group, Prinsstraat 13, 2000 Antwerp, Belgium

ARTICLE INFO

Available online 13 February 2013

Keywords:
Grasp
Path relinking
Relocate distance
Edit distance
Capacitated vehicle routing problem
Statistical analysis

ABSTRACT

In this paper we develop an intelligent path relinking procedure for the capacitated vehicle routing problem, based on the relocate distance. This procedure transforms an incumbent solution into a guiding solution in a minimal number of relocate moves. In each step of the path relinking procedure, one customer is removed from the solution and re-inserted in another position.

The path relinking procedure is integrated in a GRASP (greedy randomized adaptive search procedure) and VND (variable neighborhood descent) framework and thoroughly tested. This analysis shows that the path relinking procedure is not able to improve the performance of a simple GRASP + VND metaheuristic, but some interesting conclusions can nonetheless be drawn.

A second contribution of this paper is an analysis of the computational results based on sound statistical techniques. Such an analysis can be useful for the field of metaheuristics, where computational results are generally analyzed in an ad hoc way and often with dubious statistical validity.

1. Path relinking and the vehicle routing problem

The capacitated vehicle routing problem (CVRP) is defined on a graph $G=(V,E)$ with $V=v_0 \cup \{v_1,\ldots,v_N\}$. The set $\{v_1,\ldots,v_N\}$ represents a set of customers and v_0 represents a depot. The edges represent travel costs between customers. Each of the N customer has a non-negative known demand q_i ($i=1,\ldots,N$). This demand must be serviced by a homogeneous set of vehicles, all having capacity Q. Travel costs c_{ij} between customers i and j are known and constant. Sometimes an extra cost (called drop cost e_i) is incurred for each customer visited.

The objective of the CVRP is to determine a set of minimum-cost routes that satisfy the following two constraints:

1. Each route begins and ends at the depot.
2. The total demand serviced in a single route does not exceed the capacity Q of the vehicles.

Some authors use a variant of the CVRP in which the total cost in a single route (sum of travel costs and drop costs) is not allowed to exceed a given maximum cost C. In this paper, we only deal with problems that do not have this constraint.

The CVRP is undoubtedly one of the best-studied problems in operations research. Given the fact that the CVRP is NP-hard, only small instances can be solved to optimality [3]. Heuristics and metaheuristics are therefore often used to solve the CVRP and a large number of metaheuristics have been developed for solving this problem. A recent overview on different metaheuristics for the CVRP can be found in Szeto et al. [22]. Notable methods include the adaptive memory search procedure of Rochat and Taillard [19], that (after more than 15 years) still tops the list of best-performing approaches on the standard Christophides benchmark instances, and the evolutionary approaches of Mester and Bräysy [12] and Nagata and Bräysy [14], that outperform other approaches on (larger) instances with up to 483 customers.

Path relinking is a relatively new metaheuristic technique for combinatorial optimization, proposed by Glover (see, e.g., [7]). Path relinking attempts to find a new good solution by examining the solutions that are on a *path* from an initial (incumbent) to a final (guiding) solution. By definition, each move on the path makes the solution more different from the initiating solution and more similar to the guiding solution. Moving on the path is done by a *neighborhood operator*, just like in any local search algorithm. The technical difference with ordinary local search is that the neighborhood search *strategy* that decides which move to execute is not based on the quality of the resulting solution, but on the *distance in the solution space* between the resulting solution and the guiding solution. A move that takes the solution closer to the guiding solution will be preferred over one that takes it further away, regardless of the quality of the resulting solution. Constructing a path relinking procedure therefore requires us to

* Corresponding author. Tel.: +32 3 265 40 48.
E-mail addresses: kenneth.sorensen@ua.ac.be (K. Sörensen), patrick.schittekat@sintef.no (P. Schittekat).

0305-0548/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2013.02.005

select a move operator to use and a distance measure *in the solution space* between two solutions. The distance measure then can be used to determine whether a move brings a solution closer to the guiding solution and whether the resulting solution can be considered to be "on a path from incumbent to guiding solution".

Usually, path relinking is not used as a stand-alone solution method, but combined with other metaheuristics, such as tabu search or GRASP (see, e.g., [17,1,15,18]).

Although the capacitated vehicle routing problem (CVRP) is one of the best-known combinatorial optimization problems, path relinking approaches for this problem are few and far between. This is partially due to the fact that a VRP solution is most naturally represented as a *set of permutations of customers*, each member of the set representing a tour. Contrary to problems that have a natural binary or vector representation, it is not immediately obvious what is meant by "moving along a path from incumbent to guiding solution". To the best of our knowledge, the only application of path relinking to the CVRP is due to Ho and Gendreau [10]. Application of path relinking to other routing problems can be found in Hashimoto and Yagiura [9] (for the vehicle routing problem with time windows), Reghioui et al. [16] (for the capacitated arc routing problem with time windows), Souffriau et al. [21] (for the team orienteering problem), and El Fallahi et al. [5] (for the multi-compartment vehicle routing problem).

The overview of methods presented in Szeto et al. [22] shows that at least 29 different methods for the CVRP exist, all achieving more or less comparable performance (the method of Szeto et al. [22] itself ranks 16th). It can be argued that, notwithstanding this abundance of methods, a thorough understanding of the fundamental mechanisms that drive the performance of a metaheuristic for the CVRP is still lacking from the literature. This is at least partially the result of the fact that most authors do not perform a thorough study of the performance of the metaheuristic they present: parameters are set and components are combined through trial-and-error. Such an ad hoc analysis may result in conclusions (e.g., method X is better than method Y) that do not stand the scrutiny of statistical testing. An equally important contribution of this paper is therefore the careful statistical analysis of the path relinking operator, the other components of the metaheuristic, as well as their parameter levels. Using appropriate statistical techniques can reveal whether the addition of a component, or the changing of a parameter to another level, improves the performance of a metaheuristic in a statistically significant way or whether any perceived improvements in performance are purely due to chance.

2. Distance-based path relinking

Glover and Laguna [8] state that the path relinking algorithms should move from an initial (incumbent) to a guiding solution by progressively introducing attributes contributed by the guiding solution. In a sense, path relinking *transforms* the incumbent solution in the guiding solution, one step at a time. As mentioned above, there is no agreement in the literature as to how this should be done and permutation problems seem to be among the most difficult problems. In this paper, we use a modified version of the edit distance to determine the transformation towards the guiding solution. Using a distance measure to perform the transformation from incumbent into guiding solution is logical and can be explained if we consider scatter search. In scatter search, new solutions are often found by taking linear combinations of existing solutions (although other types of combinations are also possible). Any linear combination of two solutions lies on the "shortest path" (i.e. the straight line) connecting these two solutions. The key property is that any convex linear combination x of solutions a and b satisfies the equality that the sum of its distance to the initial and guiding solutions is equal to the distance between these two solutions

$$d(a,x)+d(x,b)=d(a,b). \tag{1}$$

For permutation problems such as the CVRP, the "shortest path" between two solutions is not a universally agreed-upon notion. Existing path relinking approaches for vehicle routing problems have therefore been based on ad hoc procedures that do not have any relationship with the neighborhood search operators used in the other parts of the algorithm. However, given a local search operator, a distance between any two solutions can be calculated, corresponding to the minimal number of moves (using the considered neighborhood operator) required to transform the first solution into the second one. A distance measure therefore gives rise to a list of moves that, when executed in a specific order, results in a set of intermediate solutions on the "shortest path" between guiding and incumbent solution. For each of the intermediate solutions, Eq. (1) holds.

For the CVRP, many different move types have been defined in the literature (swap, relocate, 2-opt, and many more). Each of these move types gives rise to an associated distance measure. Some of these distance measures are easy to calculate, whereas others are NP-hard (see [20] for an overview).

In this research, we focus on the *relocate* move. This move removes a customer from a vehicle routing solution and inserts it in a different position (see Fig. 1) in the same route or a different one.

Our path relinking procedure works in two stages. In the first stage, the *relocate distance* d_r is determined, i.e., the minimal number of relocate moves that will turn the incumbent solution into the guiding solution. In the same step, a list M containing the customers that need to be relocated in order to achieve this is determined.

In the second stage, the relocate moves themselves are performed. The customers on the list M can be moved in any order and each move will take the current solution one step closer to the guiding solution. However, the position in the solution to which each customer is moved needs to be determined carefully, in such a way that the resulting solution is indeed reached in a minimal number of moves.

In the remainder of this paper, we adopt the following conventions. We encode a vehicle routing solution as a string of symbols (integers or letters), representing the customers, using a vertical bar as the trip delimiter. It is important to note that a single vehicle routing solution can be encoded in different ways, by permuting the order of the trips or by reversing the order of the customers in a trip. This is a consequence of the fact that the routes in a vehicle routing solution are not ordered and that routes are undirected. Encodings `|123|4567|89|` and `|98|123|7654|` therefore represent the same solutions. The relocate distance is able to account for the fact that similar solutions may be encoded in many different ways, and is able to calculate the minimum number of relocate moves necessary to transform the first solution into the second *irrespective of the way both solutions have been encoded*. The relocate distance between solutions `|123|4567|89|` and `|98|123|7654|`, e.g., is zero. Since the path relinking procedure uses the relocate distance, it transforms a

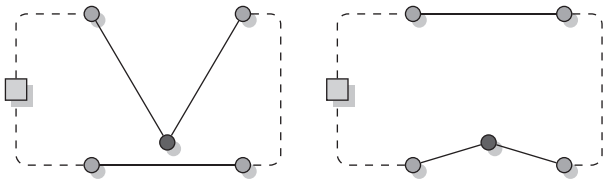

Fig. 1. Relocate move.

solution into another one in a minimal number of relocate moves, regardless of how both the initial solution and the guiding solution are encoded.

A solution $\boldsymbol{s}$ (indicated in bold) consists of one or more tours s_i, $i \in [1,\ldots,S]$, where S is the number of tours of solution $\boldsymbol{s}$. Encodings of tours or solutions are represented by the corresponding Greek letters, e.g., tour s_i can be encoded as a permutation of customers which we call σ_i. Since the vehicle routing problem is defined on an undirected graph, the reversed permutation, which we represent as $\overline{\sigma}_i$ is also a valid encoding of tour s_i. The complete solution $\boldsymbol{s}$ is encoded as a permutation of the tours, separated by the trip delimiter. Solution $\boldsymbol{s}$ is therefore encoded as $\boldsymbol{\sigma} = |\sigma_1|\sigma_2|\ldots|\sigma_S|$, $\boldsymbol{\sigma} = |\sigma_S|\overline{\sigma}_{S-1}|\ldots|\overline{\sigma}_1|$ or any other permutation.

The relocate distance between two solutions $\boldsymbol{s}$ and $\boldsymbol{t}$ of a vehicle routing problem is written as $d_r(\boldsymbol{s},\boldsymbol{t})$ and corresponds to the minimal number of relocate moves that should be performed to transform the first solution into the second. A relocate move consists of two operations: (1) the removal of a customer from the solution and (2) the insertion of this customers in another position in the solution. The number of relocate moves is therefore equal to both the number of insert operations and the number of remove operations.

2.1. Transforming a tour into another one: the remove–insert distance

For convenience, we also define the *remove–insert distance* $\delta_{ri}(\sigma_i,\tau_j)$ *between two encoded tours*. The remove–insert distance between tours σ_i and τ_j is equal to the number of operations required to transform σ_i into τ_j, where an operation is defined as either inserting a customer in the tour or removing one from it. The remove–insert distance does not require the same customers to appear in both tours.

The remove–insert distance is based on the *edit distance*, a measure to determine the difference between two strings that was developed by Wagner and Fischer [23], extending the work of Levenshtein [11]. This distance metric determines the *minimum total cost of an edit transformation that transforms a string s into a string t*. Both strings are assumed to be composed of characters of a given, finite alphabet. An edit transformation is a sequence of elementary edit operations. Three edit operations are defined: insertion of a character, deletion of a character and substitution of a character by another one. Each individual edit operation can be assigned a cost. If all costs are equal to one, the edit distance is equal to the minimum number of edit operations required to transform s into t. Wagner and Fischer [23] also provide a dynamic programming algorithm that runs in $O(n^2)$ time if both strings are approximately of length n. More efficient algorithms have been developed, but these are beyond the scope of this paper.

The remove–insert distance can be easily shown to be equal to the standard edit distance when the cost of substitutions has been set to infinity[1] and the cost of insertion and deletion to 1. As a result, it can be calculated using the same algorithms as the standard edit distance. Note that the remove–insert distance between a tour σ and an empty tour is equal to the number of customers in the tour σ. An outline of a simple dynamic programming algorithm to calculate the remove–insert distance is shown in Algorithm 1.

Algorithm 1. Calculation of remove–insert distance $\delta_{ri}(\sigma,\tau)$.

```
initialise: n ← |σ|, m ← |τ|;
if n = 0 then
    return m;
    exit;
if m = 0 then
    return n;
    exit;
Construct a matrix D of size (n+1) × (m+1);
Initialise: ∀i ∈ [0,n]:D(i,0) ← i, ∀j ∈ [0,m]:D(0,j) ← j;
for i = 1 to n do
    for j = 1 to m do
        if s1(i) = s2(j) then
            c ← 0;
        else
            c ← ∞;
        D(i,j) ← min(D(i−1,j)+1, D(i,j−1)+1, D(i−1,j−1)+c);
return D(n,m);
```

[1] In fact, any number greater than 2 will do.

The number in the bottom left cell of matrix D, i.e., $D(n,m)$ gives the remove–insert distance $\delta_{ri}(\sigma,\tau)$. The actual operations (remove customer and insert customer) that transform σ into τ can be found by tracing the path through the matrix that has been followed to find $D(n,m)$. An example clarifies this.

Example. Consider two tours `abcde` and `bcad`. The resulting matrix D, including the path from the origin cell to cell (n,m) is

	–	a	b	c	d	e
–	**0** →	**1**	2	3	4	5
		↘				
b	1	2	**1**	2	3	4
			↘			
c	2	3	2	**1**	2	3
				↓		
a	3	2	3	**2**	3	4
				↘		
d	4	3	4	3	**2** →	**3**

A horizontal move of the path indicates that a customer is removed from the first tour. A vertical move indicates that a customer is added to the first tour. A diagonal move indicates a correspondence between the customer in the row and that in the column, and therefore no need to add or remove anything. In this way, customers from the first solution that are neither added or removed can be linked to customers of the second solution in a structure called a *trace* [23]. A trace for the example under consideration is the following.

```
a       b       c               d       e
        |       |               |
        b       c       a       d
```

The trace clearly shows that customer `a` needs to be both removed and inserted in different positions and that customer `e` needs to be removed. Those three operations can be done in any order, but customers that are inserted should be done so in a correct position. To determine this position (which is not necessarily unique), the following procedure can be used. First, customers that are linked by the trace are labeled as *fixed* and put on a

list. Any customer that is inserted should be in the same position it occupies in the guiding tour *relative only to the fixed customers*. Once a customer has been inserted, it is also labeled fixed.

A possible remove–insert transformation from abcde to bcad is therefore given by the following sequence. Fixed customers in each step are underlined

$$\underline{abcde} \xrightarrow{\text{remove a}} \underline{bcde} \xrightarrow{\text{add a}} \underline{bcade} \xrightarrow{\text{remove b}} \underline{bcad}$$

When customer a is added, e.g., it needs to be added *after* customers b and c and *before* customer d because this is the position it appears in the guiding tour. Its position relative to customer e is irrelevant.

Because customers are inserted relative only to fixed customers, they can often be inserted in several positions. E.g., if customer a needs to be inserted after c and before g, which are the only fixed customers in solution bcdefgh, it can be inserted in four possible positions (i.e., immediately following c, d, e or f).

2.2. Transforming a solution into another one: the relocate distance

The relocate distance $d_r(\boldsymbol{s},\boldsymbol{t})$ between solutions $\boldsymbol{s}$ (encoded as $\boldsymbol{\sigma}$) and $\boldsymbol{t}$ (encoded as $\boldsymbol{\tau}$) can be calculated as follows:

1. Calculate the remove–insert distance $\delta_{ri}(\sigma_i,\tau_j)$ between each tour of the first solution and each tour of the second solution, as well as the remove–insert distance between each tour of the first solution and the reverse of each tour of the second solution, i.e., $\delta_{ri}(\sigma_i,\overline{\tau}_j)$.
2. Create a square matrix A containing the tours of solution $\boldsymbol{s}$ as the rows and those of solution $\boldsymbol{t}$ as the columns. If one of the solutions has fewer tours than the other one, rows or columns with empty tours are added so that the resulting matrix is a square one. Each cell contains the minimum of the two values calculated in the previous step, i.e., the cell corresponding to row s_i and column t_j contains the value
$$a_{ij} = \min[\delta_{ri}(\sigma_i,\tau_j),\delta_{ri}(\sigma_i,\overline{\tau}_j)].$$
3. Assign each route of $\boldsymbol{s}$ to a route of $\boldsymbol{t}$ by solving a minimum-cost assignment problem using the matrix A as cost matrix. The relocate distance between solutions $\boldsymbol{s}$ and $\boldsymbol{t}$ is equal to half of the cost of the assignment.

When the relocate distance has been determined, each tour of solution $\boldsymbol{s}$ is assigned to the tour of solution $\boldsymbol{t}$ that it will be transformed into. Each pair of tours has a remove–insert distance, which corresponds to a list of customers to be inserted and/or removed. When all lists are compiled into a single list, each customer on the list will appear twice: once to be removed and once to be inserted. These customers are the ones that should be relocated to transform the incumbent into the guiding solution. The order in which this is done is arbitrary (see further for a discussion), but the insert operation of the relocate move should be made relative to the fixed customers (i.e., the customers that are not to be relocated or those that have already been relocated).

Algorithm 2. Path relinking for the VRP pseudo code.

Input: Two solutions $\boldsymbol{s}$ and $\boldsymbol{t}$
Output: $n-1$ intermediate solutions $\boldsymbol{u}_1$ to $\boldsymbol{u}_{n-1}$
1 Calculate $n = d_r(\boldsymbol{s},\boldsymbol{t})$ and determine the list M of customers to move;
2 Put all other customers on the list F of fixed customers;
3 Set $\boldsymbol{u}_0 = \boldsymbol{s}$;
4 **for** $i=1$ *to* $n-1$ **do**
5 Remove customer i from M;
6 Add customer i to F;
7 Create solution $\boldsymbol{u}_i$: move customer i in $\boldsymbol{u}_{i-1}$ to the position it occupies in $\boldsymbol{t}$ relative to the customers in F;

2.3. Path relinking for the CVRP: worked-out example

Given an incumbent solution encoded as |123|4567| and a guiding solution encoded as |67|421|53| (see Fig. 2), the aim of the path relinking procedure is to transform the former into the latter in a minimal number of relocate operations. Note that the guiding solution has more tours than the incumbent solution.

The remove–insert distances between tours and reverse tours of both solutions are calculated, the results are shown in Fig. 3.

Solving an assignment problem using matrix A in Fig. 3(c) as the cost matrix assigns tours 123 to 124 (remove 3, add 4), 4567 to 67 (remove 4 and 5) and an empty tour to 35 (add 3 and 5). The total remove–insert distance is $\delta_{ri} = 6$, resulting in a relocate distance $d_r = 3$. From this assignment, the list of customers to move M is compiled. This list consists of customers 3, 4 and 5. List F of fixed customers consists of customers 1, 2, 6, and 7.

Using the rules outlined above (i.e., a customer should be moved to the position it occupies in the guiding solution relative to the fixed customers), the incumbent solution can be transformed into the guiding solution in three relocate moves. If we execute the relocate moves in the order 4, 3, 5, the resulting transformation is the one shown in Fig. 4.

2.4. Path relinking design questions

Two questions still need to be resolved. As mentioned, moves in the path relinking procedure can be executed in an arbitrary

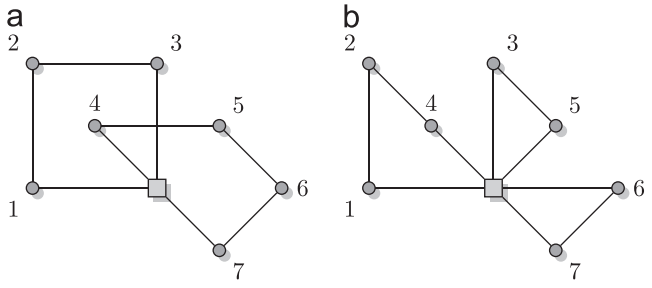

Fig. 2. Path relinking for the VRP: incumbent and guiding solution. (a) Incumbent solution |123|4567|. (b) Guiding solution |67|124|53|.

	67	421	53
123	5	4	3
4567	2	5	4
–	2	3	2

(a)

	76	124	35
123	5	2	3
4567	4	5	4
–	2	3	2

(b)

	67	124	35/53
123	5	**2**	3
4567	**2**	5	4
–	2	3	**2**

(c) Min of (a) and (b) (matrix A)

Fig. 3. Remove–insert distances required to calculate the relocate distance.

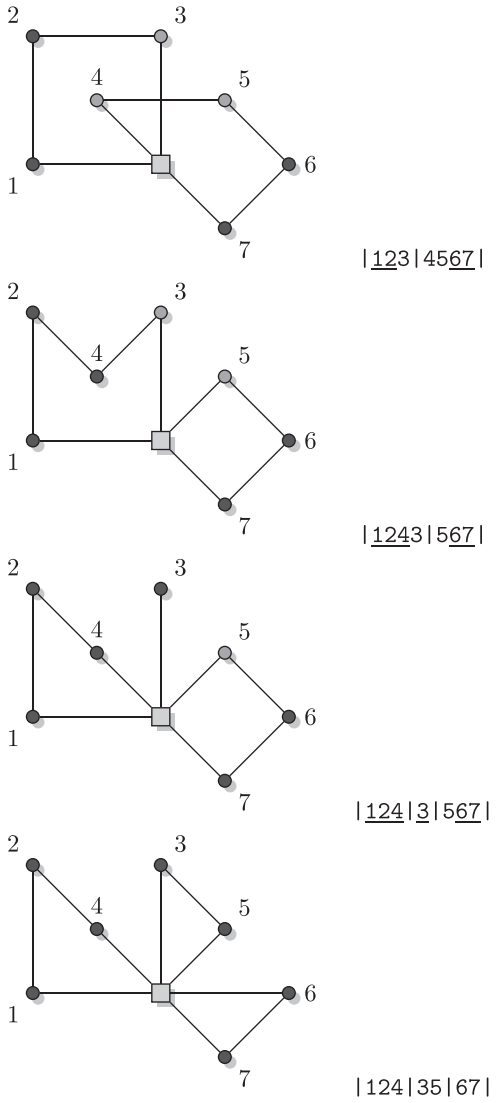

Fig. 4. Path relinking for the vehicle routing problem: worked out example. Fixed customers are darker in the figure and underlined in the encoding.

order and a *move selection strategy* is needed to determine which move to select at each step in the path relinking procedure. In this research we test two such strategies: `ms-best` and `ms-random`. The `ms-random` move selection strategy selects at each step a random customer from the list M of customers to move and relocates this customer, irrespective of whether this results in a feasible solution or not. The `ms-best` move selection strategy selects the best move at each step, and is a bit more involved. At each step of the path relinking procedure this strategy examines each possible relocate move and establishes what the cost of the resulting solution will be, whether this resulting solution is feasible and—if the solution is infeasible—how much the capacity of all trucks combined would be exceeded. If at least one relocate move leads to a feasible solution, it selects the relocate move that results in the feasible solution with the lowest cost. If none of the relocate moves lead to a feasible solution, it selects the one that results in the solution that is "least infeasible", i.e., in which the capacity is exceeded by the smallest amount.

The move selection strategy determines the relocate move to perform at each step of the path relinking procedure. Another design question is which solution the path relinking procedure should return and (as a direct consequence) how many solutions should be examined on the path from incumbent to guiding solution. We distinguish and test three *solution selection strategies*. The `ss-best` strategy searches the entire path from incumbent to guiding solution and returns the feasible solution with the lowest cost. The `ss-first` strategy stops the search as soon as a feasible solution is encountered with a lower cost than the incumbent solution. The `ss-middle` strategy chooses the feasible solution that is closest to the middle of the path. In other words, this strategy chooses the solution that has rank closest to $d_r/2$. If no feasible solution can be found on the path from guiding to incumbent solution, no solution is returned by the path relinking procedure, regardless of the solution selection strategy.

3. A GRASP+PR+VND algorithm for the CVRP

Path relinking is generally not used as a stand-alone optimizer. First of all, an initial archive of solutions should be generated for the path relinking procedure to relink. Secondly, local search optimization is necessary to achieve acceptable performance. Our path relinking procedure is therefore embedded in an algorithm that includes a *greedy randomized adaptive search procedure* (GRASP) construction phase and a *variable neighborhood descent* (VND) improvement phase. We use the term GRASP+VND to symbolize the algorithm *without* path relinking procedure and GRASP+PR+VND for the algorithm *with* path relinking procedure, whatever the order in which path relinking and variable neighborhood descent are executed. Both the GRASP and the VND are standard components of many algorithms for the CVRP and we therefore describe them briefly. First, we discuss the main structure of our GRASP+VND and GRASP+PR+VND algorithms.

3.1. GRASP+PR+VND algorithm outline

Integration of the GRASP, the PR and the VND procedure is done in the following way. First, the GRASP algorithm is allowed to generate n solutions (n is a parameter of the algorithm). These solutions are put in an archive. Optionally, the solutions in the archive are improved by VND. Then, the PR procedure takes each pair of solutions in the archive and relinks them, returning a single solution, unless no feasible solution was found on the path between guiding and incumbent solution. In this way, a maximum of $n(n-1)$ solutions are added to the archive. Again, an optional run of the VND algorithm improves every solution in the archive. The algorithm ends by returning the best solution in the archive.

Because vehicle routing algorithms that do not use local search are easily outperformed by algorithms that do use it, we only examine configurations in which the VND procedure occurs at least once. This gives rise to one configuration for the GRASP+VND algorithm and three for the GRASP+PR+VND algorithm, in which the VND procedure is executed *before* PR (`vnd-pre`), *after* PR (`vnd-post`) or both before *and* after PR (`vnd-both`).

3.2. GRASP procedure

Our GRASP procedure is based on the well known Clarke and Wright [4] insertion heuristic for the vehicle routing problem. This heuristic starts from a solution in which all customers are visited in separate routes. The heuristic then builds a *savings matrix* that, for each pair of customers, contains the decrease in cost (or "saving") that would result from connecting the customers, thereby merging the two routes that contain the customers. The saving is achieved by merging two routes with customers i and j and is calculated by adding distance c_{i0} of driving from stop i to the depot 0, and from the depot 0 to stop j minus the extra

distance or cost c_{ij} of driving from stop i to j

$$s_{ij} = c_{i0} + c_{0j} - c_{ij}. \quad (2)$$

For two customers to be "connectable", they have to appear in different routes. The CVRP is usually defined on an undirected graph, in which customers can be merged if they are currently first or last in their respective routes and the total capacity required by the two routes containing the customers is not larger than the vehicle capacity. The corresponding route merger removes the two route segments between the depot and both customers while it adds a route segment between the two customers. In each iteration, the Clarke–Wright heuristic selects the pair of customers which have to be connected to produce the largest savings.

The major weakness of the Clarke–Wright savings algorithm is that it is completely greedy and therefore often produces very effective routes in the beginning, but at the end of the search the quality of the route merges deteriorates very quickly. Gaskell [6] and Yellow [24] adapt the original Clarke–Wright savings formula by introducing a positive parameter λ and change the savings formula to

$$s_{ij} = c_{i0} + c_{0j} - \lambda c_{ij}. \quad (3)$$

In this paper, we take a different approach and introduce randomness into the solution construction process. Instead of performing the best possible route merger at each iteration, our GRASP procedure builds a *restricted candidate list* (RCL) that contains the α best route mergers (unless less than α route mergers are possible). From this list, a random route merger is selected and performed. α is a parameter of the GRASP procedure.

3.3. VND procedure

Variable neighborhood descent (VND) is a deterministic variant of the well-known variable neighborhood search (VNS) heuristic [13]. Like VNS, VND changes the neighborhood operator once the search is stuck in a local optimum, but VND differs from VNS in which no random perturbation is used. There exist many different move types for the CVRP. In our VND algorithm (as in many others), we use the following three:

1. *Relocate*, as explained before, removes a single customer from a route and inserts it into another route or in another position in the same route.
2. *Exchange* switches the positions of two customers.
3. 2-*Opt* selects two edges which are removed from the current solution. The resulting incomplete route is then reconnected by introducing two new edges. In our VND heuristic, 2-opt moves between a pair of routes can be implemented following the same principle.

The neighborhood search operators are executed in the order they appear in the list, with each neighborhood being searched in a best-improving way, i.e., all possible moves of the current neighborhood are examined and the move that results in the feasible solution with lowest cost is chosen and executed. If an improving move is found, the search restarts using the first (relocate) neighborhood. When no more moves can be found that improve the current solution using the current neighborhood, the search continues with the next neighborhood. The VND procedure ends when the current solution is a local optimum of all three neighborhoods.

4. Experiments

Notwithstanding the fact that the field of statistics offers a large array of tools that can be used to test metaheuristic methods in a scientifically valid way, appropriate techniques are rarely used in the metaheuristics literature. Rather, comparing different algorithms or different configurations of an algorithm is usually done in blatant disregard of any existing techniques that would make such comparison statistically valid. Just counting the number of times method A outperforms method B and drawing conclusions without looking at the statistical validity of claims made on this basis is not good science.

In this paper, we test our GRASP+PR+VND algorithm using an extensive statistical analysis that consists of two phases. In the first phase, we tune our GRASP and PR procedure to perform in the best possible way. In the second phase, we test whether our algorithm works better *with* path relinking than *without*. The analysis focuses on solution quality first and treats CPU time as a secondary target. In general, CPU times of all our algorithms are in line with those in the literature, i.e., several seconds up to 1 min for instances of up to 100 customers.

Prior to the analysis, we have selected several potential levels for the parameters of our GRASP+PR+VND algorithm. Table 1 shows the different parameter levels, their symbols used in the analysis, and the number of levels for each parameter.

4.1. Phase I: best parameter settings for GRASP path relinking

In this phase, we determine the best settings for the GRASP and path relinking component of our algorithm. To this end, we run a full factorial experiment on a subset of the Augerat et al. [2] instances. We selected three instances from each of the sets A, B and P: the smallest instance, the largest instance, as well as a medium-sized instance. The selected instances were `A-n32-k5.vrp`, `A-n45-k7.vrp`, `A-n80-k10.vrp`, `B-n31-k5.vrp`, `B-n50-k8.vrp`, `B-n78-k10.vrp`, `P-n16-k8.vrp`, `P-n50-k10.vrp`, and `P-n101-k4.vrp`.

For all parameter setting combinations in Table 1, we run our GRASP+PR+VND algorithm on each of the nine instances, resulting in $162 \times 9 = 1458$ runs. The GRASP+VND algorithm is run on the nine instances, with all possible parameter settings for α and n, resulting in $9 \times 9 = 81$ runs. For each run, we measure the

Table 1
Levels for the different parameters of GRASP+PR+VND.

Parameter	Symbol	Component	# levels	Values
Size of the restricted candidate list	α	GRASP	3	5, 10, 20
Number of initial solutions to generate	n	GRASP	3	5, 10, 20
PR move selection strategy	`MoveSel`	PR	2	`ms-best`, `ms-random`
PR solution selection strategy	`SolSel`	PR	3	`ss-first`, `ss-middle`, `ss-best`
When to run VND relative to PR	`VNDStrat`	VND	3	`vnd-pre`, `vnd-post`, `vnd-both`
Number of level combinations			162	

accuracy, which we define as the percentage deviation from the (known) optimal solution. We also measure the computing time.

The observations are fitted to a statistical model using a multi-way ANOVA (analysis of variance). The advantages of a multi-way ANOVA over a one-way ANOVA are that it also allows for the detection of interaction effects between two or more parameters, reduced error variance because it takes into account all explanatory variables at the same time, and, as a result, more powerful significance tests. The results of this statistical procedure tell us which parameters are important to achieve good performance and what levels of these parameters are significantly better than others. To avoid over-fitting the model to the nine investigated problems, we introduce an extra variable (the *problem* variable) that represents all nine problem instances, and is seen as a random-effect variable. In other words, we regard these nine problems as being randomly selected from a larger population.

A type-III ANOVA model is used which contains fixed effects for the parameters of the GRASP+VND or GRASP+PR+VND algorithm, as well as random effects for the nine problem instances. The dependent variable is the *accuracy* of the results. The model is built sequentially by first adding main effects and then interaction effects. During model construction, variables which represent a significant effect are withheld in the model whereas variables with an effect that is not statistically significant are excluded.

An attractive feature of multi-way ANOVA is that it is able to provide so-called least squares means (LS means), which offer an additional advantage over traditional group means, because they control the levels of the other categorical effects included in the model. To this end, other effect levels are set as much as possible to *neutral* values. As a result, LS means are more representative than traditional group means when analyzing the differences across levels of main and/or interaction effects.

The R^2 statistic, which gives an indication of the amount of variation around the mean of the dependent variable that can be explained by the model, can be augmented (increasing the quality of the model) by adding covariates to the model. Covariates are quantitative variables which may have a direct or indirect effect on the dependent variable. For this model, we have considered covariates such as problem size N, problem tightness and the coefficient of variation for the distances between each customer pair. All selected covariates can easily be obtained. Problem size equals the number of considered customers while problem tightness is defined as the ratio of total demand over the minimum number of vehicle needed multiplied by the vehicle capacity. The coefficient of variation for the distances between the customers is the ratio of the standard deviation to the mean distance between the customers. In all of the experiments, however, only the problem size ever showed any significant effect and we therefore exclude all other covariates from the discussion further in this paper.

4.1.1. Multi-way ANOVA model to determine best parameter settings for GRASP+VND

Using the multi-way ANOVA model with covariates, we obtain an R^2 value of 0.547664, indicating that the model explains approximately 54% of total variation around the mean. Table 2 shows the parameters that were found to have a statistically significant effect on the accuracy of the method.

When comparing the LS means for each main and interaction effect, we can draw the following conclusions for the GRASP+VND procedure. As can be expected, the number of solutions generated by the GRASP method n has a significant effect on the solution quality. A t-test shows that a value of $n=20$ produces significantly better solutions than a value $n=5$ (of course at the cost of an increase in computing time). The interaction between α and N (the problem size) is also significant, but the effect is such that a large problem size should be accompanied by a small value of α and vice versa. A potential explanation for this is that small problem sizes require a large amount of randomness in the GRASP heuristic to locate enough different solutions without ending up in the same basin of attraction. For large problems, a small value of α is better as this increases the amount of intensification.

4.1.2. Multi-way ANOVA model to determine best parameter settings for GRASP+PR+VND

The multi-way ANOVA model with covariates for the GRASP+PR+VND algorithm resulted in an R^2 value of 0.455719 indicating that around 45% of the variation around the mean can be explained by the model. The significant model parameters can be found in Table 3.

When comparing the LS means for each main and interaction effect, the following conclusions can be drawn for the GRASP+PR+VND procedure. All conclusions are significant according to a t-test at the 95% confidence level.

- Number of solutions generated by the GRASP heuristic: $n=20$ is significantly better than $n=10$, $n=10$ is significantly better than $n=5$.
- Randomness of the GRASP heuristic: values of $\alpha=20$ and $\alpha=10$ are significantly better than $\alpha=5$. There is no significant difference between $\alpha=10$ and $\alpha=20$.
- `SolSel`: The `ss-middle` strategy is significantly better than `ss-best` and `ss-first`.
- `VNDStrat`: `vnd-both` is significantly better than `vnd-post`, `vnd-post` is significantly better than `vnd-pre`.
- `SolSel` × `MoveSel`: The combination `ms-best-ss-middle` outperforms all other combinations in a statistically significant way.

Some of these conclusions confirm the obvious: generating more solutions also yield better solutions, enough randomness is needed for the GRASP heuristic to generate diverse initial solutions, and performing VND before ánd after PR generates the best solutions. Other conclusions are more interesting. For the path relinking procedure, there is no significant difference between the `ms-random` move selection strategy and `ms-best` move selection strategy. However, the `ss-middle` solution selection

Table 2
Multi-way ANOVA model with covariates for GRASP+VND.

Source	F ratio	Prob > F
n	3.3262	0.0419
$\alpha \times N$	7.2406	0.0014

Table 3
Multi-way ANOVA model with covariates for GRASP+PR+VND.

Source	F ratio	Prob > F
n	102.6406	< 0.0001
α	5.1411	0.0060
`SolSel`	4.6418	0.0098
`VNDStrat`	18.9353	< 0.0001
`MoveSel` × `SolSel`	5.1594	0.0059
$\alpha \times N$	41.6759	< 0.0001
`VNDStrat` × N	3.5799	0.0281

strategy significantly outperforms the other two (ss-first and ss-best). In other words, path relinking works best if the solution returned by the PR procedure is closest to the middle solution between guiding and incumbent, regardless of the quality of this solution. Another interesting conclusion is that the ss-middle solution selection strategy is best combined with a ms-best move selection strategy, even though the ms-best move selection strategy is itself not significantly better than the ms-random move selection strategy.

4.2. Phase II: contribution of path relinking

We have established in phase I that the best solution selection strategy for the path relinking procedure is ss-middle, and that this is best combined with the ms-best solution selection strategy. Also, the vnd-both strategy was shown to outperform both vnd-post and vnd-pre. In this phase, we test whether the path relinking procedure in its best configuration can outperform a simple GRASP+VND strategy. We test both ms-best and ms-random move selection strategies because these strategies are not significantly different.

To this end, we run a full factorial experiment in which we perform a run of the GRASP+VND method and a run of the GRASP+PR+VND method using all possible combinations of levels of parameters α (5, 10, 20) and n (5, 10, 20) on all 73 Augerat et al. [2] instances. Because the GRASP+PR+VND procedure with a parameter n generates a maximum of n^2 solutions in the archive, the GRASP+VND procedure is similarly allowed to generate n^2 (25, 100, 400) solutions.

We then perform paired t-tests, both on the average performance of the algorithm over the nine different GRASP settings and on the best GRASP setting. The following conclusions can be drawn.

When averaged over all parameter combinations of α and n, the average accuracy is very similar for GRASP+VND (0.02064), GRASP+PR+VND using ms-best (0.02212), and GRASP+PR+VND using ms-random (0.02609). Paired t-tests show that GRASP+VND significantly outperforms GRASP+PR+VND using ms-best, which is in turn significantly better than GRASP+PR+VND using ms-random. In other words, when averaged over all possible GRASP parameter settings, a simple GRASP+VND outperforms both GRASP+PR+VND methods.

If we compare for each of the 73 instances the result of GRASP+VND and GRASP+PR+VND using the best combination of α and n for each method *and for each instance*, we again obtain a very similar average accuracy for GRASP+VND (0.01112), GRASP+PR+VND using ms-best (0.01047), and GRASP+PR+VND using ms-random (0.01134). None of these differences are statistically significant. In other words, there is no difference in performance between GRASP+VND and GRASP+PR+VND if we consider only the best settings for the GRASP initial construction phase.

Finally, comparing the best single parameter setting across all instances for GRASP+VND ($\alpha=10$ and $n=20$, average accuracy=0.01444), for GRASP+PR+VND using ms-best($\alpha=10$ and $n=20$, average accuracy=0.01456) and GRASP+PR+VND using ms-random ($\alpha=20$ and $n=20$, average accuracy=0.01739). The difference between GRASP+VND and GRASP+PR+VND using ms-best is not significant, but both are significantly better than GRASP+PR+VND using ms-random. In other words, the best possible setting for GRASP+PR+VND is not able to outperform GRASP+VND.

From the above results, we cannot conclude that the path relinking procedure improves the GRASP+VND algorithm in a statistically significant way. Given the fact that the PR procedure is relatively complicated, these findings give very little incentive to use this procedure in a real-life environment. The GRASP+PR+VND algorithm was usually somewhat faster than the GRASP+VND algorithm, but this difference was insignificant and increasing the computing time of both methods usually resulted in a very similar increase in quality. In an additional experiment, we compared the GRASP algorithm without path relinking to the GRASP+PR algorithm (using the best settings for both the GRASP and the PR procedure). Again, both methods were allowed to generate the same number of solutions (25, 100 or 400). Although significantly faster, the GRASP+PR algorithm was on average about 5% worse than the simple GRASP method.

A potential explanation for the fact that the path relinking procedure does not improve the GRASP+VND procedure is that both algorithms find very close-to-optimal solutions, on average about 1% from optimal, and that for these problems there is just very little improvement possible. On the other hand, if the path relinking procedure would have been significantly better, then this would have been detected by the fact that the GRASP+PR+VND algorithm would have found equally good solutions for smaller values of n. Another possibility is that the GRASP+VND procedure is so powerful that outperforming it is near impossible. The fact that almost all algorithms in the list of best-performing algorithms for the CVRP use strategies that are similar (especially the extensive use of local search using different neighborhood structures is extremely common) more or less supports this claim.

5. Conclusions and future research

This paper has presented a novel path relinking procedure for the capacitated vehicle routing problem. This procedure is based on the relocate distance and transforms an incumbent solution into a guiding solution in a minimal number of relocate moves. Such a procedure is much closer to the true spirit of path relinking than any other method proposed so far.

Unfortunately, closeness to the spirit of path relinking is no guarantee for success, and despite extensive parameterization of our procedure, we could not show that it can significantly improve the performance of a simple GRASP+VND procedure. Nonetheless, some interesting conclusions can be drawn from our experiments. Most importantly, it is surprising to see that the strategy to choose the solution closest to the center of the path generated by the path relinking procedure outperforms both other tested strategies, including the one that selects the best solution on the path. This shows that path relinking can indeed find solutions that have characteristics of both guiding and incumbent solutions and that this can be a useful diversification strategy.

It is also worth noting that we have tested our method in a much more statistically rigorous way than is common in the metaheuristics literature, using a carefully designed experiment analyzed by a multi-way ANOVA method. We firmly believe that this way of determining the best possible parameter setting and establishing which components of a heuristic contribute to the solution quality and which not is far superior and should always be preferred over more traditional methods of comparison. Just counting the number of best-known solutions found, or the average deviation from the best-known solution (using parameters set on a per-instance basis), we could probably have claimed that our method was "among the best-performing approaches". Instead, we have chosen to only report the more robust conclusion that our GRASP+PR+VND is as powerful as GRASP+VND but not more.

Future research on this topic will focus around the use of distance-based path relinking for other problems, as well as improving our path relinking procedure for the CVRP (e.g., by allowing it to make more than a single move type). The use of adequate statistical techniques for testing metaheuristics is another topic that we intend to explore further. For example, we have always used a full factorial design in this paper, but

fractional factorial designs are potentially more efficient, especially if the number of parameters or parameter levels is large.

References

[1] Alfandari L, Plateau A, Tolla P. A path relinking algorithm for the generalized assignment problem. In: Resende MGC, De Sousa JP, editors. Metaheuristics: computer decision-making. Boston: Kluwer; 2004. p. 1–18.

[2] Augerat P, Belenguer JM, Benavent E, Corberán A, Naddef D, Rinaldi G. Computational results with a branch and cut code for the capacitated vehicle routing problem. In: Rapport de recherche-IMAG, 1995. ISSN 0750-7380.

[3] Baldacci R, Toth P, Vigo D. Exact algorithms for routing problems under vehicle capacity constraints. Annals of Operations Research 2010;175(1):213–45.

[4] Clarke G, Wright JW. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 1964;12(4):568–81.

[5] El Fallahi A, Prins C, Wolfler Calvo R. A memetic algorithm and a tabu search for the multi-compartment vehicle routing problem. Computers & Operations Research 2008;35(5):1725–41.

[6] Gaskell TJ. Bases for vehicle fleet scheduling. Operations Research 1967;18(3):281–95.

[7] Glover F. Scatter search and path relinking. In: Corne D, Dorigo M, Glover F, editors. New ideas in optimization. McGraw–Hill; 1999.

[8] Glover F, Laguna M. Fundamentals of scatter search and path relinking. Control and Cybernetics 2000;29(3):653–84.

[9] Hashimoto H, Yagiura M. A path relinking approach with an adaptive mechanism to control parameters for the vehicle routing problem with time windows. In: Van Hemert J, Cotta C, editors. Evolutionary computation in combinatorial optimization. Springer; 2008. p. 254–65.

[10] Ho Sin C, Gendreau Michel. Path relinking for the vehicle routing problem. In: Proceedings of nordic MPS 2004. The ninth meeting of the nordic section of the mathematical programming society, Norrköping, Sweden, 2004.

[11] Levenshtein VI. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics—Doklady 1966;10:707–10.

[12] Mester D, Bräysy O. Active-guided evolution strategies for large-scale capacitated vehicle routing problems. Computers & Operations Research 2007;34:2964–75.

[13] Mladenović N, Hansen P. Variable neighborhood search. Computers & Operations Research 1997;24(11):1097–100.

[14] Nagata Y, Bräysy O. Edge assembly-based memetic algorithm for the capacitated vehicle routing problem. Networks 2009;54:205–15.

[15] Oliveira CAS, Pardalos PM, Resende MGC. GRASP with path-relinking for the quadratic assignment problem. Experimental and efficient algorithms. Springer; 2004. p. 356–68.

[16] Reghioui M, Prins C, Labadi N. GRASP with path relinking for the capacitated arc routing problem with time windows. Applications of evolutionary computing. Springer; 2007. p. 722–31.

[17] Resende MGC, Ribeiro CC. GRASP and path relinking: recent advances and applications. Technical Report. AT&T Labs Research; 6 April 2003.

[18] Resende MGC, Werneck RF. A hybrid multistart heuristic for the uncapacitated facility location problem. Technical Report TD-5RELRR. AT&T Labs Research; 2003.

[19] Rochat Y, Taillard ÉD. Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics 1995;1(1):147–67.

[20] Sörensen K, Reimann M, Prins C. Permutation distance measures for memetic algorithms with population management. In: Proceedings of MIC'2005, the sixth metaheuristics international conference, Vienna, Austria, 2005.

[21] Souffriau W, Vansteenwegen P, Vanden Berghe G, Van Oudheusden D. A path relinking approach for the team orienteering problem. Computers & Operations Research 2010;37:1853–59.

[22] Szeto WY, Wu Y, Ho SC. An artificial bee colony algorithm for the capacitated vehicle routing problem. European Journal of Operational Research 2011;215(1):126–35.

[23] Wagner RA, Fischer MJ. The string-to-string correction problem. Journal of the ACM 1974;21:168–73.

[24] Yellow PC. A computational modification to the savings method of vehicle scheduling. Operational Research Quarterly 1970;21(2):281–3.