# An Innovative Metaheuristic Solution Approach for the Vehicle Routing Problem with Backhauls

Emmanouil E. Zachariadis, Chris T. Kiranoudis

Department of Process Analysis and Plant Design, National Technical University of Athens, Athens, Greece, {ezach@mail.ntua.gr, kyr@chemeng.ntua.gr}

**Abstract**

This paper deals with a practical transportation model known as the Vehicle Routing Problem with Backhauls (VRPB), which aims at designing the minimum cost route set for satisfying both delivery and pick-up demands. In methodological terms, we propose a local search metaheuristic which explores rich solution neighborhoods composed of exchanges of variable-length customer sequences. To efficiently investigate these rich solution neighborhoods, tentative local search move are statically encoded by data structures stored in Fibonacci Heaps which are special priority queue structures offering fast minimum retrieval, insertion and deletion capabilities. To avoid cycling phenomena and induce diversification, we introduce the concept of promises, which is a parameter-free mechanism for coordinating the conducted search. The proposed metaheuristic development was tested on well-known VRPB benchmark instances. It exhibited fine performance, as it consistently generated the best-known solutions for all the examined benchmark problems.

***keywords***: metaheuristics, vehicle routing, computational complexity

## 1. Introduction

The distribution of goods is a central operational process lying at the heart of modern business activity. It constitutes a great proportion of a company's total running costs. For this reason great scientific interest has been dedicated to the development of effective solution approaches for optimising real-life logistics operations.

The most widely studied problem model in the context of logistics optimisation is the classical Vehicle Routing Problem (VRP). VRP aims at generating the minimum cost set of routes for a homogeneous fleet of vehicles based at a central depot. The generated routes originate and terminate at the central depot, and they must satisfy the product demand of a given customer population which is assumed to be fixed and known in advance. Each customer must be visited by a single vehicle only once. In addition, the carrying load of a vehicle cannot exceed its capacity. Based on the aforementioned classical VRP version, researchers have proposed and examined several VRP variants that capture the special requirements of practical logistics processes. One of these problem variants is the VRP with backhauls (VRPB) which involves both delivery and pick-up demands.

Briefly, the VRPB aims at designing the optimal routes to satisfy the delivery and pick-up demand of linehaul and backhaul customers, respectively. It models the following scenario: Each vehicle departs from the depot and is initially unloaded by satisfying the linehaul demand. After the load of the vehicle has been exhausted, it visits the backhaul customer where goods are again loaded onto the vehicle to be delivered back to central depot. Consequently, the load of the vehicle monotonically decreases, as goods are delivered to the linehaul customers, and reaches to zero after the last delivery customer has been visited. Then, backhauls are serviced causing the vehicle load to monotonically increase before returning back to the central station. The precedence constraint which forces linehauls to be serviced before backhauls is imposed to the problem model due to the fact that "*the vehicles are rear-loaded and rearrangement of the loads on the trucks at the delivery points is not deemed feasible.*" (Goetschalckx and Jacobsblecha ,1989).

In graph theoretic terms, the VRPB model is defined on a complete graph $G = (V, A)$ where $V = \{v_0\} \cup L \cup B$ is the vertex set and $A$ is the edge set. Sets $L = \{v_1, v_2, \ldots, v_l\}$ and $B = \{v_{l+1}, v_{l+2}, \ldots, v_{l+b}\}$ denote the linehaul and backhaul customer sets, respectively, whereas vertex $v_0$ corresponds to the central depot which acts as the station of $K$ vehicles with capacity $Q$. With each linehaul customer $v_i \in L$ is associated a delivery product quantity $d_i$ which must be transported from the depot to the customer, while with each backhaul customer $v_j \in B$ is associated a pick-up quantity $p_j$ which must be shipped from the customer back to the central station. With each arc $(v_i, v_j) \in A$ is associated a fixed non-negative cost $c_{ij}$ which reflects the cost involved for traveling from location $v_i$ to $v_j$. The goal of the VRPB is to design a set of routes such that:

*a)* The size of the generated route set is equal to $K$.

*b)* Every customer is assigned to exactly one route.

*c)* Every route contains at least one linehaul customer (no empty routes are allowed, no routes servicing only backhaul customers are allowed)

*d)* Within every route, linehaul customers precede backhaul customers.

*e)* The total delivery demand of the linehaul customers assigned to a route does not exceed vehicle capacity $Q$.

*f)* The total pick-up demand of the backhaul customers assigned to a route does not exceed vehicle capacity $Q$.

*g)* The total cost of the generated route set is minimised.

The interested reader is referred to the works of Goetschalckx and Jacobsblecha (1989), Toth and Vigo (1997) and Mignozzi *et al* (1999) for mathematical formulations of the VRPB model.

Our interest in the VRPB is motivated both by its great practical and theoretical importance. From the commercial viewpoint, VRPB is frequently encountered by large companies who must transport goods from their production site to the retailer outlets (linehauls), while at the same time the production site must be supplied from vendors (backhauls) located within the same geographic region (Goetschalckx and Jacobsblecha, 1989). In addition, pro-environmental practices raise the necessity of bi-directional product flows modeled by VRPB. Products are transported from the production site to the retailers, while at the same time used and outdated products are collected from the retailers and sent back to the production site in order to be recycled, disassembled or appropriately processed before being disposed. From the theoretical perspective, VRPB is a significantly challenging optimisation problem. It reduces to the classical VRP when only linehaul customers are considered ($B = \emptyset$). Thus, being a generalisation of the VRP, the VRPB variant is an NP-hard combinatorial optimisation problem.

The purpose of the present paper is to propose an original metaheuristic methodology to solve the VRPB. The proposed local-search metaheuristic algorithm explores rich solution neighborhoods, exchanging variable-length customer sequences instead of performing single customer swaps and relocations. This is efficiently achieved by statically encoding the tentative local search moves using the Static Move Descriptor entities (Zachariadis and Kiranoudis 2009a). To induce diversification and eliminate cycling phenomena, we introduce the concept of *promises*, which is a parameter-free mechanism for coordinating the progress of the overall local-search method. Our VRPB metaheuristic was successfully tested on well-known benchmark instances, consistently producing high-quality solutions.

The remainder of the present article is organized as follows: Section 2 provides a literature review on previous VRPB solution approaches, followed by Section 3 which presents in detail the proposed solution methodology. The computational results are summarized in Section 4. Finally, Section 5 concludes the paper.

## 2. Literature Review

As previously stated, VRPB is an NP-hard combinatorial optimisation problem. Thus, exact solution methodologies are able to solve rather small-scale instances within acceptable computational times. On the contrary, to deal with medium- and large-scale practical VRPB instances, researchers have concentrated on the design of heuristic and metaheuristic solution approaches, which do not guarantee optimality but are computationally manageable.

In terms of VRPB heuristic procedures, Deif and Bodin (1984) propose two solution methodologies by modifying the Clarke and Wright (1964) heuristic originally designed for the VRP. The first method imposes a constraint which forces deliveries to occur before any pick-up services begin. For the second approach, this precedence constraint is guaranteed by incorporating a penalty factor in the savings function. Goetschalckx and Jacobsblecha (1989) propose a methodology that constructs a good-quality initial solution by the application of spacefilling curve heuristics. The final solution is generated by means of an improvement algorithm. Goetschalckx and Jacobsblecha (1993) propose a cluster-first, route-second VRPB algorithm based on the generalised assignment approach of Fisher and Jaikumar (1981). Another VRPB algorithm that belongs to the cluster-first, route-second category of heuristics has been presented by Toth and Vigo (1996). Their algorithm is based on a K-tree Lagrangian relaxation presented for the VRP (Fisher

1994). Toth and Vigo (1999) propose another cluster-first route-second algorithm for solving both the symmetric and asymmetric VRPB. Their approach exploits information included in infeasible solutions associated with a lower-bound produced by using a Lagrangian approach described in the study of Toth and Vigo (1997).

Regarding more recent metaheuristic strategies, Osman and Wassan (2002) propose a two-phase VRPB methodology. In the first phase the initial solution is produced by two construction heuristics based on the saving-insertion and saving-assignment procedures, respectively. The solution is then improved by a reactive Tabu Search (TS) which considers single-node and two-node exchange neighborhood structures. The reactive concept is used to control the balance between the intensification and diversification of the search. Another TS based algorithm has been proposed by Wassan (2007). The former work is a hybridization of TS and Adaptive Memory Programming (AMP). The proposed Adaptive Memory drives the conducted search towards unexplored solution regions. Brandão (2006) presents a tabu search scheme for improving the initial solution which is produced by two different procedures. The first way of generating the initial solution is to solve two distinct Open VRP (OVRP) subproblems, one for the linehaul and one for the backhaul customers. The other approach of building the initial solution consists of obtaining a pseudo-lower bound by making Lagrangian relaxations, so that the routing problem is transformed into a minimum K-tree problem. The proposed TS procedure examines three neighborhood structures that involve relocating a customer to another route, exchanging two customers belonging to two different routes, and exchanging the positions of a linehaul and a backhaul customer within the same route. Ropke and Pisinger (2006) present a general algorithmic framework which effectively deals with numerous routing variants that consider backhaul customers. Their approach is based on Large Neighborhood Search (Shaw, 1998). Finally, Gajpal and Abad (2009) present an ant colony VRPB metaheuristic which makes use of two multi-route local search schemes.

Except for the above-presented heuristic and metaheuristic solution approaches, researchers have also proposed exact methodologies for the VRPB. The first such work is due to Yano *et al* (1987). Their methodology solves a practical VRPB application using customized route generation routines combined with a branch-and-bound procedure. Toth and Vigo (1997) present a branch-and-bound algorithm for the VRPB. To derive the lower bound on the optimal solution cost, they propose a Lagrangian relaxation of some problem constraints. To strengthen the Lagrangian bound, valid inequalities are added in a cutting plane fashion. Finally, Mignozzi *et al* (1999) present a new VRPB integer programming formulation. They compute a valid lower bound to the optimal solution via the combination of different heuristic methods that deal with the dual of the LP-relaxation of the integer programming model. Their proposed branch-and-bound algorithm managed to optimally solve problems of up to 100 customers.

## 3. The Proposed Algorithm

As mentioned in the introductory Section of the present article, the proposed VRPB metaheuristic makes use of the Static Move Descriptor (SMD) strategy in order to reduce the computational complexity required for examining very large solution neighborhoods. To avoid being trapped in premature local optima and effectively diversify the search, we introduce a parameter-free algorithmic concept called *promises*. In this Section, we thoroughly present the aforementioned algorithmic components and later discuss on the overall metaheuristic development.

### 3.1 The Solution Neighborhoods and their SMD representation

Instead of single vertex exchanges and relocations, the proposed methodology explores a rich neighborhood structure consisting of every possible exchange of vertex sequences (thereafter called *bones*) that involve from 0 to $\mu$ customers (Zachariadis and Kiranoudis 2009b). Let Variable Length Bone Exchange (VLBE) denote the aforementioned neighborhood structure. Except for the VLBE operator, our methodology also examines the classical 2-opt local-search operator.

#### 3.1.1 The VLBE local search operator

The computational complexity required for exhaustively investigating the VLBE solution neighborhood is obviously bounded by O($n^2\,\mu^2$), as there are $n^2$ vertex pair combinations, and $\mu^2$ are the 2-combinations of the two bone lengths (customers contained in the bones exchanged). For practical problem instances of significant $n$ values, the $O(n^2\ \mu^2)$ complexity of the VLBE move type would lead to excessive computational times.

To efficiently explore the VLBE neighborhood structure, we make use of the SMD entities which encode tentative moves in a static (solution independent) manner. In specific, every VLBE SMD instance includes the following static information: a pair of node values ($n_1$ and $n_2$), and a pair of bone length values (*n1_len* and *n2_len*). The move represented by a VLBE SMD with $n_1$ = A, $n_2$ = B, *n1_len* = $a$, and *n2_len* = $b$ is the exchange of the bone beginning after node A and containing $a$ customers and the bone beginning after B and containing $b$ customers. Note that in the case where *n1_len* or *n2_len* is equal to 0; the SMD encodes a bone relocation move rather than an exchange one. Apart from the aforementioned information, every SMD instance contains a cost tag which corresponds to actual cost involved for performing the encoded move to the candidate solution. Obviously, the cost tag dynamically changes through the search process, as it depends on the structure of the current solution.

To exhaustively map the VLBE neighborhood using the SMD representation, in total $((n+K)!/(2!(n+K-2)!)) \cdot ((\mu+1)^2-1)$ VLBE SMD instances are required, where $K$ denotes the routes present in the current solution. The first term corresponds to the 2-combinations without repetition of the $n$ customers and $K$ depot vertex occurrences, whereas the second term corresponds to the 2-combinations of the bone length values that vary from 0 to $\mu$.

Figure 1 illustrates the application of three example VLBE SMD instances to a VRPB solution of eight customers and two routes.

### 3.1.2 The 2-opt local search operator

The 2-opt operator removes two edges present in the candidate solution and replaces them with a new edge pair. If the 2-opt operator is applied within a route, two edges are deleted and two new edges are generated by reversing the route path lying between the deleted edges. When the 2-opt move is implemented between a route pair, the two routes involved are divided into their starting and terminating segments by removing two solution edges. Two edges are created so that the starting segment of the first route is connected to the terminating segment of the second one, and the beginning part of the second route is linked to the terminating part of the first one. Exhaustively examining the 2-opt neighborhood structure requires O($n^2$) complexity, as each vertex pair uniquely defines a particular 2-opt move.

To encode the 2-opt local search move into SMD entities, we create one SMD instance for each vertex pair. Thus, each 2-opt SMD instance contains two node values, namely $n_1$ and $n_2$. The mechanism of applying a 2-opt SMD with $n_1$ = A and $n_2$= B is the following: If vertices A and B belong to the same route (and without loss of generality, assume that A precedes B in the route vector), A is connected to B by reversing the path beginning after A and terminating at B. Otherwise, let $rt_A$ and $rt_B$ denote the routes containing A and B, respectively. The starting route segment of $rt_A$ terminating at node A is connected to the $rt_B$ segment initiating after vertex B and terminating at the depot. Similarly, the starting segment of $rt_B$ which terminates at B is linked to the $rt_A$ segment that begins after vertex A and ends at the depot. Apart from the $n_1$ and $n_2$ values, with each 2-opt SMD instance is associated a cost tag which is equal to the cost involved for applying the encoded move to the candidate solution.

As earlier stated, each vertex pair uniquely defines a particular 2-opt move. Thus, to exhaustively represent the 2-opt neighborhood structure in total $(n + K)! / (2!(n + K- 2)!)$ SMD instances are required, corresponding to the 2-combinations without repetition of the $n$ customers and $K$ depot vertex occurrences.

Figure 2 provides three example applications of 2-opt SMD instances to a VRPB solution of eight customers and two routes.

### 3.2 Updating the cost tags of the SMD instances

As earlier explained, the SMD instances statically encode the tentative local search moves defined by the neighborhoods structures. In addition, they include a cost label (*cst*) which is equal to the actual cost required for implementing the encoded move to a candidate solution. This cost label is obviously dynamic in the sense that it depends on the particular structure of a VRPB solution. Thus, as local search moves are applied to the candidate solution, the cost tags of the SMD instances must be appropriately updated in order to be valid according the modified solution states. The main advantage of the SMD representation of local search moves comes from the fact that when a local search move is applied to a given solution, only a limited part of the solution structure is modified. Therefore, to keep the SMD instances updated, only the cost tags of the SMD instance subset which is associated with the modified solution part have to be reevaluated. In the following, we provide the rules that determine the SMD instance subset which has to be updated when either a VLBE or a 2-opt SMD instance is applied to the candidate solution.

To facilitate exposition, for any VRPB solution, we introduce the following notation:

- *pred*($v$) denotes the bone that contains (up to) $\mu$ vertices and terminates before vertex $v$
- *bone*($v$, $a$) denotes the bone that initiates after vertex $v$ and contains $a$ customers.
- *succ*($v$, $a$) denotes the vertex which is located $a$ positions after vertex $v$ in the vector of the route visiting $v$.
- *part*($v$, $y$) denotes the bone originating after vertex $v$ and terminating at vertex $y$.
- *init*($v$) denotes the vertex set contained in the route segment initiating from the depot and terminating before vertex $v$
- *fin*($v$) denotes the vertex set contained in the route segment initiating after vertex $v$ and terminating at the depot.

### 3.2.1 Update rules for the application of a VLBE SMD instance

Consider that a VLBE instance with $n_1$ = A, $n_2$ = B, *n1_len* = $a$, and *n2_len* = $b$ is applied to a candidate VRPB solution. The cost tags of following groups of SMD instances must be reevaluated according to the modified solution state:

1. The VLBE SMD instances with $n_1$ or $n_2$ contained in the vertex set {{A}, {B}, {*succ*(A, $a$)}, {*succ*(B, $b$)}}, corresponding to $O(\mu^2 n)$ updates.

2. The VLBE SMD instances with $n_1$ or $n_2$ contained in {*bone*(A, $a$-1), *bone*(B, $b$-1)} and relevant bone lengths referring to the route segments lying after the bones exchanged. The number of vertices contained in {*bone*(A, $a$-1), *bone*(B, $b$-1)} is $O(\mu)$, thus the necessary cost updates are bounded by $O(\mu^3 n)$.

3. The VLBE SMD instances with $n_1$ or $n_2$ contained in {*pred*(A), *pred*(B)}and relevant bone lengths that refer to the bones exchanged. At most $O(\mu)$ vertices are contained in {*pred*(A), *pred*(B)}, thus at most $O(\mu^3 n)$ VLBE cost tags need to be reevaluated.

4. The 2-opt SMD instances with $n_1$ or $n_2$ contained in the set {{A}, {B}, {*succ*(A, *a*)}, {*succ*(A, *a*)}}, corresponding to $O(n)$ necessary cost updates.

5. The 2-opt SMD instances with their one node value included in {*bone*(A, *a*-1), *bone*(B, *b*-1)}, and their other node value contained in the vertex set {*init*(A), *init*(B), *fin*(succ(A, *a*)), *fin*(*succ*(B, *b*))}. The necessary updates for the aforementioned 2-opt SMD instances are bounded by $O(\mu\, n)$, as at most $O(\mu)$ nodes are contained in the two bones exchanged, and up to $O(n)$ vertices are contained in the initial and terminating segments of the routes involved in the move.

### 3.2.2 Update rules for the application of a 2-opt SMD instance

The SMD instances that must be re-evaluated when applying a 2-opt move depend on whether the move was applied within a route or between a route pair.

If an intra-route 2-opt SMD instance with $n_1$ = A, $n_2$ = B is applied to a candidate VRPB solution (without loss of generality, assume that A precedes B in the route vector), the cost tags of the following SMD instances must be updated:

1. The VLBE SMD instances with $n_1$ or $n_2$ included in *pred*(A) and relevant bone lengths that refer into the *part*(A, B) route segment which is reversed, corresponding to $O(\mu^3\, n)$ necessary cost updates.

2. The VLBE SMD instances with $n_1$ or $n_2$ contained within {{A}, *part*(A, B)}. The necessary cost updates are bounded by $O(\mu^2\, z\, n)$, where $z$ denotes the number of vertices contained in *part*(A, B).

3. The 2-opt SMD instances with $n_1$ or $n_2$ contained in {{A}, *part*(A, B)}, corresponding to $O(z\, n)$, where $z$ is the number of vertices contained in *part*(A, B).

If an inter-route 2-opt SMD instance with $n_1$ = A, $n_2$ = B is applied to a candidate VRPB solution, the cost tags of the following SMD instances must be updated:

1. The VLBE SMD instances with $n_1$ or $n_2$ contained in the vertex set {{A}, {B}}, corresponding to $O(\mu^2\, n)$ updates.

2. The VLBE SMD instances with $n_1$ or $n_2$ contained in the vertex sets *pred*(A) and *pred*(B) and relevant bone lengths that refer after vertices A, and B, respectively. The size of this SMD subset is bounded by $O(\mu^3\, n)$.

3. The 2-opt SMD instances with $n_1$ or $n_2$ contained in the set {{A}, {B}}, corresponding to $O(n)$ necessary cost updates.

4. The 2-opt SMD instances with one node value ($n_1$ or $n_2$) contained in *init*(A) and the other node value included in {*fin*(A), *fin*(B)}. In addition, the cost tag of every 2-opt SMD instance with one node value contained in the vertex set *init*(B), and the other node value included in {*fin*(A), *fin*(B)}. These necessary updates are at most $O(z_A\ z_B)$, where $z_A$ and $z_B$ denote the total number of vertices visited by the routes servicing nodes A and B, respectively.

### 3.3. The promises concept

As will be later presented, the proposed local search method implements the lowest-cost tentative moves of the examined neighborhood structures. This deterministic criterion of moving to subsequent solutions causes cycling phenomena to occur. To avoid these phenomena, we propose the concept of promises which filters out a subset of tentative moves so that the overall local search method escapes from premature local optima. The basic advantage of the proposed promises scheme is that unlike several metaheuristic strategies (Tabu Search, Guided Local Search, and Simulated Annealing), it does not require any parametric decisions and tuning. In other words, it has a flexible and robust structure which does not depend on problem-specific characteristics.

The basic rationale of the promises concept is the following: when a local search move is applied to a candidate solution *S*, some solution attributes are removed and some new solution attributes are created to form a new solution *S΄*. The eliminated attributes of *S* are stored together with a cost tag equal to the objective function value of solution *S*. Tentative moves that re-create these solution attributes at a higher cost than their cost tags are disregarded during future neighborhood evaluations. Loosely speaking, as the local search evolves, it gives a promise to every attribute that is eliminated from the candidate solution. This promise is straightforward: "*eliminated solution attributes are going to be recreated in a solution of higher-quality than the one they were last contained in*". By fulfilling these promises, the search is drastically diversified and driven towards unexplored solution space regions. Another important algorithmic characteristic is that the attribute cost tags do not monotonically increase: consider that an attribute *A* is removed from a candidate solution *S*, and is stored together with the solution cost $z(S)$. Then, it is recreated forming a solution $S'$ of cost $z(S') < z(S)$. If deteriorating structural modifications are applied to solution attributes other than *A*, the search may reach to solution $S''$ (containing *A*) of cost $z(S'') > z(S)$. Then, if a local search operator is applied to $S''$ to eliminate *A*, the cost tag of A is set equal to $z(S'')$ which is greater than its previous cost tag $z(S)$. This backtracking behavior is crucial, as it eliminates the risk of over-restricting the search by making promises which are very difficult to be fulfilled.

### 3.4. The proposed adaptation of the promises concept for the VRPB

For the proposed VRPB metaheuristic, we have selected complete routes to be the solution attributes examined. This selection proved to be effective for the test problems under consideration that contained rather low $n$ / $K$ ratios (few customers per route). On the contrary, for routing problems which involve many customers per route, the aforementioned selection would be inappropriate: cycling would be avoided, however the search would not be able to intensify into promising solution space regions, as eliminated routes would be very difficult to be re-created into a lower objective function solution. In these cases, a different attribute selection (for instance sequences of consecutive vertices) is required to achieve a balanced algorithmic behavior.

#### 3.4.1. Making promises

When an intra-route move is applied to route *rt* which belongs to a VRPB solution of cost $z$, route *rt* is associated with a cost label $tag_{rt}$ equal to $z$. Analogously, if an inter-route move is applied to a pair of routes *rt1* and *rt2* contained in a VRPB solution of cost $z$, the aforementioned routes *rt1* and *rt2* are associated with cost labels $tag_{rt1}$ and $tag_{rt2}$, respectively, both of them equal to $z$.

#### 3.4.2. Checking promises

A tentative intra-route move that leads to the creation of route *rt* that belongs to a VRPB solution of cost $z$ is considered, if and only if $z < tag_{rt}$. Similarly, a tentative inter-route move which leads to the generation of *rt1* and *rt2* that belong to a VRPB solution of cost $z$ is acceptable, if and only if $z < tag_{rt1}$ and $z < tag_{rt2}$.

### 3.5 The overall metaheuristic framework

The proposed VRPB metaheuristic, entitled Route Promise Algorithm (RPA) is initiated by the application of a construction heuristic algorithm, which is aimed at generating a set of feasible VRPB routes which is going to be later improved by the core of the RPA improvement method.

#### 3.5.1. Obtaining an initial set of feasible VRPB routes

To obtain an initial VRPB solution, we apply a construction method based on the Paessens (1988) heuristic for the VRP. In specific, the savings function used is: $s(v_i, v_j) = c_{i0} + c_{0j} - g \cdot c_{ij} + f \cdot | c_{i0} - c_{0j} |$, where $f$ and $g$ are stochastically generated within [0, 1] and (0, 3], respectively. To satisfy the special precedence constraints imposed by the VRPB model which force backhauls to be serviced after linehaul customers, we set $c_{ij}$ = M, for every $v_i \in L$ and $v_j \in B$, where M is greater than the most expensive of the arcs contained in set $A$. Furthermore, to ensure that no route consists of backhauls only, we consider that the cost $c_{0j}$ = M, for every $v_j \in B$ (Brandão, 2006). Regarding the carrying load of the vehicles, insertion positions are only considered if they do not cause any capacity constraint violation. When a (linehaul) customer is assigned to an empty route, a new empty route is generated and becomes available for subsequent customers. The construction method is terminated after every customer is assigned to a route.

#### 3.5.2. Managing the fleet size

The route set generated by the construction method described in 3.5.1 satisfies both the precedence and capacity constrains of VRPB. However, the size of the generated route set $Kcons$ is not necessarily equal to $K$. Three cases may arise: if $Kcons = K$, the proposed improvement method is executed by setting the cost $c_{00}$ (for every depot vertex occurrence) equal to M, so that no route is eliminated during the search process. If $Kcons < K$, ($K$ - $Kcons$) new empty routes are generated and inserted into the VRPB route set. Again, the cost $c_{00}$ is set equal to M, so that customers are forced into the empty routes and the final solutions consist of exactly $K$ non-empty routes. Finally, if $Kcons > K$, we set the cost $c_{0j} = c_{0j}$ + M for every customer vertex $v_j \in L \cup B$. Having used the aforementioned penalization policy, when the proposed RPA method initiates, it is primarily aimed at eliminating any depot-adjacent arcs, or in other words targets to remove any unnecessary routes. If during the course of the RPA search, the non-empty routes become equal to *K*, the penalized costs of depot-adjacent arcs are restored to their original values, and the cost $c_{00}$ (for every depot vertex occurrence) is set to M, so that no further route is removed from the VRPB candidate solution during the search progress. As will be later indicated in the Computational Results, for all test problems, the proposed scheme of managing the total number of routes succeeded on producing solutions consisting of exactly *K* non-empty VRPB routes.

### 3.5.3. The core of the proposed VRPB metaheuristic

After the initial set of VRPB routes is generated by the construction heuristic of 3.5.1, the proposed RPA metaheuristic is applied. The SMD instances for the VLBE and 2-opt neighborhood structures are generated and inserted into the Fibonacci heaps $FH_{VLBE}$ and $FH_{2\text{-}opt}$, respectively. To reduce the total number of generated SMD instances, so that the algorithm is accelerated, we use the following strategy which filters out SMD instances which are highly unlikely to represent cost improving local search moves (Tarantilis *et al*, 2008). With each vertex $v_i$, we calculate the $avg_i$ cost equal to the average cost of every arc adjacent to $v_i$ in the arc set *A*. Vertex $v_i$ is associated to its neighboring vertex set $NV_i$ which contains every vertex $v_j$ such that $c_{ij} < avg_i$. To create a (VLBE or 2-opt) SMD instance with $n_1$ = A and $n_2$= B, one of the following must hold: $\mathrm{A} \in NV_{\mathrm{B}}$ and $\mathrm{B} \in NV_{\mathrm{A}}$, A or B is the depot vertex.

After all SMD instances have been generated and stored into the Fibonacci Heaps, the iterative core of the RPA method initiates: At each iteration, the minimum-cost, feasible, and admissible (in terms of the promises concept) VLBE and 2-opt SMD instances are retrieved from the corresponding Fibonacci Heaps. The lowest cost of these two SMD instances is applied to the current solution *S*, if it is cost improving. Otherwise, if both SMD instances deteriorate the solution score, the VLBE SMD instance is selected to be applied to *S*. Before the selected SMD instance is applied, the affected route(s) are stored into a hashtable structure together with the cost of solution *S*. The selected SMD is applied to obtain the new candidate solution *S′*, and the cost tags of the affected SMD instances are updated according to the modified state of solution *S′*, following the update rules presented in 3.2. Then solution $S$ is set equal to $S'$ for the subsequent RPA iteration. The RPA method terminates when a certain time bound has been reached. The pseudocode of the RPA metaheuristic is presented in Table 1.

Note that the cost matrix modifications presented in 3.5.1 and 3.5.2, for dealing with the precedence and fixed-fleet requirements, are considered throughout the whole RPA execution.

#### 3.5.4. Computational complexity of the proposed algorithm

In this paragraph, we discuss on the computational complexity required by each of the RPA steps, as they are presented in the pseudocode of Table 1.

In terms of the initialization phase contained in Lines 5 and 6, it is performed once in $O(\mu^2 n^2)$, as the population of SMD instances is bounded by $O(\mu^2 n^2)$, and for each SMD instance, the cost tag evaluation and Fibonacci Heap insertion is performed in constant time.

To select the SMD instance to be implemented in the candidate solution (Lines 9 and 10), the following procedure is iteratively executed until the particular SMD which is both feasible and promise-keeping is identified: The minimum-cost SMD instance is retrieved from the corresponding Fibonacci Heap in O(1), and its feasibility is checked in constant time. To do so, we use the feasibility investigation approach also used for the Open VRP (Zachariadis and Kiranoudis, 2009b). However, for the VRPB model which differentiates between linehaul and backhaul customers, with every vertex $v_i$, we make use of two demand metrics, responsible for storing the total delivery and pick-up demands of all the $v_i$ predecessors contained in its route. Finally, to check whether an SMD instance is promise-keeping or not, the following operations are performed: the keys (string representations) of the tentative routes are prepared in $O(n\_rt)$, where $n\_rt$ denotes the number of customers assigned to these tentative routes. Then the cost tags are retrieved from the hashtable, and the comparisons presented in 3.4.2 are executed in O(1). Thus, the total computational complexity required for investigating the feasibility and promise-keeping status of a tentative SMD instance is bounded by the complexity required for generating the string representation of the tentative routes. Regarding the total number of SMD instances required to be retrieved from the Fibonacci Heaps until the one to be implemented is identified, it depends on both the instance characteristics (hardness of the capacity constraints), as well as the state of the candidate solution. Experimental runs indicated however, that the computational time dedicated to the move selection process is insignificant compared to the time required by the necessary cost updates (Line 19). The complexity required for the move implementation and promise making operations (Lines 16-18) is O(*n_rt*), where *n_rt* is the number of customers assigned to the affected routes. The most effort consuming task of a complete RPA iteration (Lines 8-22) is the cost tag update process of Line 19: a detailed discussion on the number of the necessary updates is given in 3.2. Each of these cost updates involves three steps: deleting the SMD instance from the heap, evaluating the new cost tag, inserting the SMD instance back to the heap. The deletion step requires O(log *m*), where *m* denotes the number of SMD instances stored in the heap, while the cost tag evaluation and insertion steps are both executed in O(1).

## 4. Computational Results

To assess the effectiveness of the proposed method, we have applied it to the VRPB benchmark instances introduced by Goetschalckx and Jacobs-Blecha (1989). The RPA solution approach was coded in Visual C# and executed on a single core of an Intel T5500 (1.66GHz). The aforementioned benchmark problems and the solutions obtained are available at [http://users.ntua.gr/ezach/](http://users.ntua.gr/ezach/).

### 4.1 Benchmark Instances

To test the effectiveness of the proposed method, we have solved the VRPB instance set generated and introduced by Goetschalckx and Jacobs-Blecha (1989). It consists of 62 problem instances in total. For all 62 instances, the depot is located at (12000, 16000), whereas the x- and y- customer coordinates are stochastically taken from [0, 24000] and [0, 32000], respectively. The customer demand is generated from a normal distribution with a mean and standard deviation equal to 500 and 200 product units, respectively. The details of the VRPB data set of Goetschalckx and Jacobs-Blecha (1989) are presented in Table 2.

To obtain the cost $c_{ij}$ of an arc ($v_i$, $v_j$), researchers have followed two different schemes: Under the first scheme - also used in our work - the Euclidean distance between a vertex pair is calculated using double precision, and then it is multiplied by 10. The product is then rounded to the nearest integer to obtain the arc cost. For this scheme of arc cost evaluation, the cost of the final solution is divided by 10 and then rounded to the nearest integer value. Under the second scheme, the cost matrix is obtained as the Euclidean distances without rounding but the solution score is rounded to two decimal places. These two distinct ways of obtaining the VRPB cost matrix make algorithmic comparison a little problematic. However, the deviation between the solution scores evaluated with the use of the two above-mentioned schemes is rather small, so that the obtained solution scores can be securely compared.

### 4.2 Parameter Setting

As seen from the detailed presentation of the proposed metaheuristic, the promises concept which constitutes the core of the search strategy does not contain any parameters. Thus, complex parameter tuning experiments were avoided before executing the RPA method. The only parameter that had to be fixed is the bone length $\mu$ which determines the number of vertices in the sequences considered by the VLBE local search operator. Obviously, the setting of $\mu$ depends on the number of customers per route which is an instance-specific characteristic. For the 62 VRPB test problems, the $n$ / $K$ ratio varies from 3.1 to 21.4, averaging at 11.9. Following the computational experience of the VLBE operator applied to OVRP instances of similar customer per route ratios (Zachariadis and Kiranoudis, 2009b), we set $\mu$ = 6, for problems with $n$ / $K \geq 6$. For the other problems (with $n$ / $K < 6$), we set $\mu = \lceil n / K \rceil$.

Regarding the termination condition used for a single RPA execution, it was set to the completion of 300 CPU seconds for problems with $n \geq 50$, and 120 CPU seconds for instances involving up to 50 vertices.

### 4.3 Computational Results on the VRPB instances

To test the effectiveness of the robustness of the RPA method, we executed it 10 times to solve each of the 62 VRPB test problems. Note that although the RPA search is deterministic, the final solution was not necessarily the same for all 10 executions, because each run involved a different initial VRPB solution (the $f$ and $g$ savings parameters of the construction heuristic are randomly generated). The results obtained are summarized in Table 3. RPA exhibited a rather robust behavior, as for 23 test instances, the same final solution was obtained for all ten algorithmic executions. The percent gap between the best and the average solution score over the ten runs obtained for each VRPB instance varied from 0.000% and 1.586%, averaging at a satisfactory 0.378%. In terms of the fixed fleet vehicle requirement, the proposed methodology consistently generated VRPB solutions of $K$ non-empty routes, for every test problem, and for all ten runs. Regarding the computational time elapsed before the overall best solutions were generated, it varies from 2 seconds for the 25-customer instances A2 and A3 to 172 seconds for the 150-customer problem N4.

To compare the RPA performance against that of the most effective VRPB metaheuristics previously published, we provide Table 4. In specific the best solution values obtained by the RPA method are presented together with those obtained by the Tabu Search of Brandão (TS) (2006), the LNS heuristic of Ropke and Pisinger (LNS) (2006), and the Multi Ant Colony System (MACS) proposed by Gajpal and Abad (2009). Note that for the three aforementioned methods, we provide the solution scores obtained with their standard parameter setting, so that a fair comparison is conducted. For the TS and LNS methods, we present the *K-tree_r* and *6R-no learning* algorithmic versions, respectively, which, on average, yielded the best results. The solution scores presented for the TS, LNS and MACS approaches are the best ones obtained after five, ten and eight algorithmic runs, respectively. To facilitate comparisons, we have rounded the reported results to integer values, because researchers have used different schemes for obtaining the VRPB cost matrix (see 4.1).

The computational time reported for the LNS and MACS methods is the average time required for a complete run. For the TS method, it is the average time required by the runs which achieved the best solutions. On the contrary, for the proposed RPA metaheuristic, we provide the average (over the 62 problems) CPU time elapsed, when the best solutions were encountered during the best of the ten RPA executions (column $t_{bst}$ of Table 2).

As seen from Table 3, the RPA method consistently matched the best-known solution scores for all 62 benchmark instances. In terms of the best reported standard algorithmic scores, the proposed metaheuristic reached higher-quality solutions for six test problems. Regarding the computational time required by the compared algorithms, we do not intend to make a detailed comparison, as the algorithmic speed depends on various factors which cannot be securely compared: processors, programming languages, compilers, memory frequencies, programming skills, etc. Furthermore, the RPA method involved a fixed time bound, whereas the TS, LNS and MACS approaches were executed for a fixed number of iterations.

At this point, we should note that for the five test problems (F1, F2, F3, F4, and L1) marked with an asterisk in Table 2, two discrepant instance versions seem to have been studied in the VRPB literature: in specific, for the four problems of group F, the website of Prof. Marc Goetschalckx provides two different instance sources. The first one suggests that the demand of the linehaul customer lying at (x, y) = (5103, 11065) is 101 (used for obtaining the RPA results reported in Table 3), whereas the second one sets this customer demand equal to 1013 product units. For problem L1, although both instance sources set the vehicle capacity equal to 4000, the works of Brandão (2006) and Gajpal and Abad (2009) suggest that the capacity is equal to 4400 product units (used for obtaining the RPA results in Table 3). To test the RPA method for both instance versions, we solved the F1, F2, F3, F4, and L1 VRPB instances, setting the demand of the F-group customer at (x, y) = (5103, 11065) equal to 1013, and the vehicle capacity of instance L1 equal to 4000. As seen from the results, all of the best solutions scores exactly match the ones obtained by the LNS method (small deviations are due to different rounding schemes used), implying that Ropke and Pisinger (2006) have considered the instance versions presented in Table 4.

## 5. Conclusions

In the present paper we have proposed a local-search algorithm for the VRPB. The proposed metaheuristic method has the ability of efficiently examining very rich solution neighborhoods by statically encoding local search moves using the SMD representation (Zachariadis and Kiranoudis, 2009a). To avoid cycling phenomena and induce diversification, we introduce the concept of promises which is a parameter-free mechanism for coordinating the solution space exploration. Briefly, the promises concept can be summarized as follows: solution attributes which are eliminated by a local search move applied to a solution of cost $z$, can only be recreated by a local search move that leads to a solution of cost $z' < z$. The proposed methodology was applied to 62 well-known VRPB benchmark instances, exhibiting fine performance. In specific, it managed to match the best-known solution scores for all 62 test problems.

Regarding future research directions, the promises concept can be tested to combinatorial optimisation problems taking under consideration various solution attributes. These tests can be easily performed, as the promises mechanism does not require time-consuming parametric tuning experiments.

**References**

Brandão J (2006). A new tabu search algorithm for the vehicle routing problem with backhauls. Eur J Opl Res 173: 540–555

Clarke G and Wright J W (1964). Scheduling of vehicle from a central depot to a number of delivery points. Opns Res 1964; 12:568 –581.

Deif I and Bodin L (1984). Extension of the Clarke and Wright algorithm for solving the vehicle routing problem with backhauling. In Kidder A (ed). Proceedings of the Babson Conference on Software Uses in Transportation and Logistic Management 1984, pp 75–96.

Fisher M (1994). Optimal solution of vehicle routing problems using minimum k-trees. Opns Res 42: 626–642.

Fisher M L and Jaikumar R (1981). A generalised assignment heuristic for vehicle routing. Networks 11: 109 –124.

Goetschalckx M and Jacobs-Blecha C (1989). The vehicle routing problem with Backhauls. Eur J Opl Res 42: 39-51.

Goetschalckx M and Jacobs-Blecha C (1993). The vehicle routing problem with backhauls: properties and solution algorithms. Technical Report MHRC-TR-88-13, Georgia Institute of Technology.

Mingozzi A, Giorgi S, and Baldacci R (1999). An exact method for the vehicle routing problem with backhauls. Transportation Sci 33: 315–329.

Osman I H and Wassan N A (2002). A reactive tabu search meta-heuristic for the vehicle routing problem with back-hauls. J Sched 5: 263–285.

Paessens H (1988). The savings algorithm for the vehicle routing problem. Eur J Opl Res 34: 336–344.

Ropke S and Pisinger D (2006). A unified heuristic for a large class of vehicle routing problems with backhauls. Eur J Opl Res 171: 750-775.

Shaw P (1998). Using constraint programming and local search methods to solve vehicle routing problems. Proceedings CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming).

Tarantilis C D, Zachariadis E E and Kiranoudis C T (2008). A Hybrid Guided Local Search for the Vehicle-Routing Problem with Intermediate Replenishment Facilities. INFORMS J Comput 20: 154-168.

Toth P and Vigo D (1996). A heuristic algorithm for the vehicle routing problem with backhauls. In: Bianco L and Toth P (eds). Advanced Methods in Transportation Analysis 1996. Springer, pp. 585–608.

Toth P and Vigo D (1997). An exact algorithm for the vehicle routing problem with backhauls. Transportation Sci 31: 372–385.

Toth P and Vigo D (1999). A heuristic algorithm for the symmetric and asymmetric vehicle routing problem with backhauls. Eur J Opl Res 113: 528 –543.

Wassan N A (2007). Reactive tabu adaptive memory programming search for the vehicle routing problem with backhauls. J Opl Res Soc 58: 1630–1641.

Yano C, Chan T, Richter L, Cutler T, Murty K and McGettigan D (1987). Vehicle routing at quality stores. Interfaces 17: 52–63.

Yuvraj G and Abad P L (2009). Multi-ant colony system (MACS) for a vehicle routing problem with backhauls. Eur J Opl Res 196: 102-117.

Zachariadis E E and Kiranoudis C T (2009a). A Strategy for Reducing the Computational Complexity of Local Search-Based Methods, and its Application to the Vehicle Routing Problem. Technical Report. National Technical University of Athens. (http://users.ntua.gr/ezach/).

Zachariadis E E and Kiranoudis C T (2009b). An Open Vehicle Routing Problem metaheuristic for examining wide solution neighborhoods. Technical Report. National Technical University of Athens. (http://users.ntua.gr/ezach/).

# TABLES

Table 1. Pseudocode of the proposed RPS metaheuristic

```
VRPB Solution RPS (VRPB Solution S_0)
    Fibonacci Heap FH_VLBE, FH_2-OPT
    VRPB Solution S, S′, S*
    HashTable RoutePromises
    SMD app, VLBE SMD vlbe, 2-opt SMD 2opt

    -- initialization phase
    generate every VLBE SMD instances according to S_0 and store them in FH_VLBE
    generate every 2-opt SMD instances according to S_0 and store them in FH_2-opt
    set S = S_0, S* = S_0

    -- improvement phase
    while (termination condition = false)
        -- local search move selection
        vlbe = lowest-cost, feasible, and promise-keeping VLBE SMD instance stored in FH_VLBE
        2opt = lowest-cost, feasible, and promise keeping 2-opt SMD instance stored in FH_2-OPT
        if ( z(S_0) + vlbe_cst < z(S*) OR z(S_0) + 2opt_cst < z(S*) )
            if (vlbe_cst < 2opt_cst) app = vlbe else app = 2opt end if
        else
            app = vlbe
        end if

        -- local search move application
        let rt1 (and rt2) denote the route (routes) affected by the move encoded by SMD app
        store the rt1 (rt2) route key (keys) together with the value z(S) into RoutePromises
        implement the SMD app to S to obtain S′
        apply the update rules (3.2) for the affected VLBE and 2-opt SMD instances according to S′
        set S = S′

        if (z(S) < z(S*)) S* = S end if
    end while
    return S*
```

Table 2. VRPB benchmark instances characteristics

| **Instance** | ***n*** | ***line*** | ***back*** | ***K*** | ***Q*** | ***cpr*** | **Instance** | ***n*** | ***line*** | ***back*** | ***K*** | ***Q*** | ***cpr*** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A1 | 26 | 20 | 5 | 8 | 1550 | 3.1 | H4 | 69 | 45 | 23 | 5 | 6100 | 13.6 |
| A2 | 26 | 20 | 5 | 5 | 2550 | 5.0 | H5 | 69 | 45 | 23 | 4 | 7100 | 17.0 |
| A3 | 26 | 20 | 5 | 4 | 4050 | 6.3 | H6 | 69 | 45 | 23 | 5 | 7100 | 13.6 |
| A4 | 26 | 20 | 5 | 3 | 4050 | 8.3 | I1 | 91 | 45 | 45 | 10 | 3000 | 9.0 |
| B1 | 31 | 20 | 10 | 7 | 1600 | 4.3 | I2 | 91 | 45 | 45 | 7 | 4000 | 12.9 |
| B2 | 31 | 20 | 10 | 5 | 2600 | 6.0 | I3 | 91 | 45 | 45 | 5 | 5700 | 18.0 |
| B3 | 31 | 20 | 10 | 3 | 4000 | 10.0 | I4 | 91 | 45 | 45 | 6 | 5700 | 15.0 |
| C1 | 41 | 20 | 20 | 7 | 1800 | 5.7 | I5 | 91 | 45 | 45 | 7 | 5700 | 12.9 |
| C2 | 41 | 20 | 20 | 5 | 2600 | 8.0 | J1 | 95 | 75 | 19 | 10 | 4400 | 9.4 |
| C3 | 41 | 20 | 20 | 5 | 4150 | 8.0 | J2 | 95 | 75 | 19 | 8 | 5600 | 11.8 |
| C4 | 41 | 20 | 20 | 4 | 4150 | 10.0 | J3 | 95 | 75 | 19 | 6 | 8200 | 15.7 |
| D1 | 39 | 30 | 8 | 12 | 1700 | 3.2 | J4 | 95 | 75 | 19 | 7 | 6600 | 13.4 |
| D2 | 39 | 30 | 8 | 11 | 1700 | 3.5 | K1 | 114 | 75 | 38 | 10 | 4100 | 11.3 |
| D3 | 39 | 30 | 8 | 7 | 2750 | 5.4 | K2 | 114 | 75 | 38 | 8 | 5200 | 14.1 |
| D4 | 39 | 30 | 8 | 5 | 4075 | 7.6 | K3 | 114 | 75 | 38 | 9 | 5200 | 12.6 |
| E1 | 46 | 30 | 15 | 7 | 2650 | 6.4 | K4 | 114 | 75 | 38 | 7 | 6200 | 16.1 |
| E2 | 46 | 30 | 15 | 4 | 4300 | 11.3 | L1 | 151 | 75 | 75 | 10 | 4400 | 15.0 |
| E3 | 46 | 30 | 15 | 4 | 5225 | 11.3 | L2 | 151 | 75 | 75 | 8 | 5000 | 18.8 |
| F1 | 61 | 30 | 30 | 6 | 3000 | 10.0 | L3 | 151 | 75 | 75 | 9 | 5000 | 16.7 |
| F2 | 61 | 30 | 30 | 7 | 3000 | 8.6 | L4 | 151 | 75 | 75 | 7 | 6000 | 21.4 |
| F3 | 61 | 30 | 30 | 5 | 4400 | 12.0 | L5 | 151 | 75 | 75 | 8 | 6000 | 18.8 |
| F4 | 61 | 30 | 30 | 4 | 5500 | 15.0 | M1 | 126 | 100 | 25 | 11 | 5200 | 11.4 |
| G1 | 58 | 45 | 12 | 10 | 2700 | 5.7 | M2 | 126 | 100 | 25 | 10 | 5200 | 12.5 |
| G2 | 58 | 45 | 12 | 6 | 4300 | 9.5 | M3 | 126 | 100 | 25 | 9 | 6200 | 13.9 |
| G3 | 58 | 45 | 12 | 5 | 5300 | 11.4 | M4 | 126 | 100 | 25 | 7 | 8000 | 17.9 |
| G4 | 58 | 45 | 12 | 6 | 5300 | 9.5 | N1 | 151 | 100 | 50 | 11 | 5700 | 13.6 |
| G5 | 58 | 45 | 12 | 5 | 6400 | 11.4 | N2 | 151 | 100 | 50 | 10 | 5700 | 15.0 |
| G6 | 58 | 45 | 12 | 4 | 8000 | 14.3 | N3 | 151 | 100 | 50 | 9 | 6600 | 16.7 |
| H1 | 69 | 45 | 23 | 6 | 4000 | 11.3 | N4 | 151 | 100 | 50 | 10 | 6600 | 15.0 |
| H2 | 69 | 45 | 23 | 5 | 5100 | 13.6 | N5 | 151 | 100 | 50 | 7 | 8500 | 21.4 |
| H3 | 69 | 45 | 23 | 4 | 6100 | 17.0 | N6 | 151 | 100 | 50 | 8 | 8500 | 18.8 |

***n***: Number of depot and customer vertices, ***line***: number of linehaul customers, ***back***: number of backhaul customers, ***K***: number of vehicles, ***Q***: vehicle capacity, ***cpr***: customers per route (= $n$-1/$K$)

Table 3. RPA results for the Goetschalckx & Jacobs-Blecha VRPB benchmark instances

| Instance | n | K | avg | | | bst | | | %gap | tot_t |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | $z_{avg}$ | $K_{avg}$ | $t_{avg}$ | $z_{bst}$ | $K_{bst}$ | $t_{bst}$ | | |
| A1 | 26 | 8 | 229,886 | 8.0 | 5 | 229,886 | 8 | 4 | 0.000 | 120 |
| A2 | 26 | 5 | 180,119 | 5.0 | 3 | 180,119 | 5 | 2 | 0.000 | 120 |
| A3 | 26 | 4 | 163,405 | 4.0 | 2 | 163,405 | 4 | 2 | 0.000 | 120 |
| A4 | 26 | 3 | 155,796 | 3.0 | 3 | 155,796 | 3 | 3 | 0.000 | 120 |
| B1 | 31 | 7 | 239,080 | 7.0 | 14 | 239,080 | 7 | 14 | 0.000 | 120 |
| B2 | 31 | 5 | 198,048 | 5.0 | 12 | 198,048 | 5 | 11 | 0.000 | 120 |
| B3 | 31 | 3 | 169,372 | 3.0 | 5 | 169,372 | 3 | 3 | 0.000 | 120 |
| C1 | 41 | 7 | 250,556 | 7.0 | 22 | 250,556 | 7 | 19 | 0.000 | 120 |
| C2 | 41 | 5 | 215,020 | 5.0 | 20 | 215,020 | 5 | 18 | 0.000 | 120 |
| C3 | 41 | 5 | 199,346 | 5.0 | 12 | 199,346 | 5 | 10 | 0.000 | 120 |
| C4 | 41 | 4 | 195,366 | 4.0 | 9 | 195,366 | 4 | 8 | 0.000 | 120 |
| D1 | 39 | 12 | 322,530 | 12.0 | 18 | 322,530 | 12 | 10 | 0.000 | 120 |
| D2 | 39 | 11 | 316,708 | 11.0 | 15 | 316,708 | 11 | 8 | 0.000 | 120 |
| D3 | 39 | 7 | 239,479 | 7.0 | 10 | 239,479 | 7 | 8 | 0.000 | 120 |
| D4 | 39 | 5 | 205,832 | 5.0 | 11 | 205,832 | 5 | 8 | 0.000 | 120 |
| E1 | 46 | 7 | 238,880 | 7.0 | 27 | 238,880 | 7 | 26 | 0.000 | 120 |
| E2 | 46 | 4 | 212,263 | 4.0 | 20 | 212,263 | 4 | 12 | 0.000 | 120 |
| E3 | 46 | 4 | 206,659 | 4.0 | 22 | 206,659 | 4 | 14 | 0.000 | 120 |
| F1 | 61 | 6 | 263,274 | 6.0 | 34 | 263,173 | 6 | 22 | 0.038 | 300 |
| F2 | 61 | 7 | 265,655 | 7.0 | 36 | 265,213 | 7 | 30 | 0.166 | 300 |
| F3 | 61 | 5 | 241,120 | 5.0 | 28 | 241,120 | 5 | 22 | 0.000 | 300 |
| F4 | 61 | 4 | 234,604 | 4.0 | 32 | 233,861 | 4 | 26 | 0.317 | 300 |
| G1 | 58 | 10 | 306,980 | 10.0 | 38 | 306,306 | 10 | 24 | 0.220 | 300 |
| G2 | 58 | 6 | 245,441 | 6.0 | 32 | 245,441 | 6 | 23 | 0.000 | 300 |
| G3 | 58 | 5 | 229,968 | 5.0 | 31 | 229,507 | 5 | 34 | 0.201 | 300 |
| G4 | 58 | 6 | 232,521 | 6.0 | 38 | 232,521 | 6 | 32 | 0.000 | 300 |
| G5 | 58 | 5 | 222,872 | 5.0 | 32 | 221,730 | 5 | 29 | 0.512 | 300 |
| G6 | 58 | 4 | 214,381 | 4.0 | 25 | 213,457 | 4 | 32 | 0.431 | 300 |
| H1 | 69 | 6 | 270,056 | 6.0 | 38 | 268,933 | 6 | 33 | 0.416 | 300 |
| H2 | 69 | 5 | 253,910 | 5.0 | 35 | 253,365 | 5 | 30 | 0.215 | 300 |
| H3 | 69 | 4 | 247,449 | 4.0 | 36 | 247,449 | 4 | 22 | 0.000 | 300 |
| H4 | 69 | 5 | 251,094 | 5.0 | 40 | 250,221 | 5 | 33 | 0.348 | 300 |
| H5 | 69 | 4 | 246,121 | 4.0 | 35 | 246,121 | 4 | 19 | 0.000 | 300 |
| H6 | 69 | 5 | 250,060 | 5.0 | 42 | 249,135 | 5 | 44 | 0.370 | 300 |
| I1 | 91 | 10 | 351,082 | 10.0 | 67 | 350,246 | 10 | 51 | 0.238 | 300 |
| I2 | 91 | 7 | 309,979 | 7.0 | 61 | 309,944 | 7 | 57 | 0.011 | 300 |
| I3 | 91 | 5 | 294,790 | 5.0 | 55 | 294,507 | 5 | 62 | 0.096 | 300 |
| I4 | 91 | 6 | 297,910 | 6.0 | 66 | 295,988 | 6 | 50 | 0.645 | 300 |
| I5 | 91 | 7 | 303,485 | 7.0 | 72 | 301,236 | 7 | 66 | 0.741 | 300 |
| J1 | 95 | 10 | 335,780 | 10.0 | 97 | 335,006 | 10 | 83 | 0.231 | 300 |
| J2 | 95 | 8 | 312,509 | 8.0 | 90 | 310,417 | 8 | 96 | 0.669 | 300 |
| J3 | 95 | 6 | 280,433 | 6.0 | 78 | 279,219 | 6 | 70 | 0.433 | 300 |
| J4 | 95 | 7 | 298,322 | 7.0 | 88 | 296,533 | 7 | 82 | 0.600 | 300 |
| K1 | 114 | 10 | 397,382 | 10.0 | 101 | 394,071 | 10 | 117 | 0.833 | 300 |
| K2 | 114 | 8 | 365,458 | 8.0 | 82 | 362,130 | 8 | 72 | 0.911 | 300 |
| K3 | 114 | 9 | 369,436 | 9.0 | 90 | 365,694 | 9 | 102 | 1.013 | 300 |
| K4 | 114 | 7 | 349,717 | 7.0 | 87 | 348,950 | 7 | 79 | 0.219 | 300 |
| L1 | 151 | 10 | 421,683 | 10.0 | 165 | 417,896 | 10 | 142 | 0.898 | 300 |
| L2 | 151 | 8 | 405,199 | 8.0 | 136 | 401,228 | 8 | 121 | 0.980 | 300 |
| L3 | 151 | 9 | 405,756 | 9.0 | 173 | 402,678 | 9 | 167 | 0.759 | 300 |
| L4 | 151 | 7 | 388,142 | 7.0 | 140 | 384,636 | 7 | 129 | 0.903 | 300 |
| L5 | 151 | 8 | 390,458 | 8.0 | 154 | 387,565 | 8 | 130 | 0.741 | 300 |
| M1 | 126 | 11 | 400,499 | 11.0 | 132 | 398,593 | 11 | 144 | 0.476 | 300 |
| M2 | 126 | 10 | 401,914 | 10.0 | 120 | 396,917 | 10 | 106 | 1.243 | 300 |
| M3 | 126 | 9 | 378,073 | 9.0 | 106 | 375,696 | 9 | 95 | 0.629 | 300 |
| M4 | 126 | 7 | 352,030 | 7.0 | 94 | 348,140 | 7 | 88 | 1.105 | 300 |
| N1 | 151 | 11 | 411,722 | 11.0 | 192 | 408,101 | 11 | 152 | 0.880 | 300 |
| N2 | 151 | 10 | 412,311 | 10.0 | 169 | 408,066 | 10 | 138 | 1.029 | 300 |
| N3 | 151 | 9 | 398,760 | 9.0 | 144 | 394,338 | 9 | 152 | 1.109 | 300 |
| N4 | 151 | 10 | 396,159 | 10.0 | 196 | 394,788 | 10 | 172 | 0.346 | 300 |
| N5 | 151 | 7 | 376,895 | 7.0 | 152 | 373,476 | 7 | 161 | 0.907 | 300 |
| N6 | 151 | 8 | 379,784 | 8.0 | 170 | 373,759 | 8 | 145 | 1.586 | 300 |

**avg**: average values over the ten RPA executions, **bst**: values for the run which yielded the highest quality solution, ***z***: objective function value, ***t***: time elapsed when the best solution was generated through the search, ***tot_t***: total time required for one complete RPA execution, ***%gap***: percent gap between the average and the best solution scores obtained (= $100 \cdot (z_{avg} - z_{bst}) / z_{avg}$)

Table 4. Comparative results of the best performing metaheuristic methods for the VRPB

| Instance | TS | LNS | MACS | BAS | *%gap*$_{BAS}$ | RPA | BKS |
|---|---|---|---|---|---|---|---|
| A1 | **229,886** | **229,886** | **229,886** | 229,886 | 0.000 | **229,886** | 229,886 |
| A2 | **180,119** | **180,119** | **180,119** | 180,119 | 0.000 | **180,119** | 180,119 |
| A3 | **163,405** | **163,405** | **163,405** | 163,405 | 0.000 | **163,405** | 163,405 |
| A4 | **155,796** | **155,796** | **155,796** | 155,796 | 0.000 | **155,796** | 155,796 |
| B1 | **239,080** | **239,080** | **239,080** | 239,080 | 0.000 | **239,080** | 239,080 |
| B2 | **198,048** | **198,048** | **198,048** | 198,048 | 0.000 | **198,048** | 198,048 |
| B3 | **169,372** | **169,372** | **169,372** | 169,372 | 0.000 | **169,372** | 169,372 |
| C1 | **250,557** | **250,557** | **250,557** | 250,557 | 0.000 | **250,556** | 250,556 |
| C2 | **215,020** | **215,020** | **215,020** | 215,020 | 0.000 | **215,020** | 215,020 |
| C3 | **199,346** | **199,346** | **199,346** | 199,346 | 0.000 | **199,346** | 199,346 |
| C4 | **195,366** | **195,367** | **195,367** | 195,366 | 0.000 | **195,366** | 195,366 |
| D1 | **322,530** | **322,530** | **322,530** | 322,530 | 0.000 | **322,530** | 322,530 |
| D2 | **316,709** | **316,709** | **316,709** | 316,709 | 0.000 | **316,708** | 316,708 |
| D3 | **239,479** | **239,479** | **239,479** | 239,479 | 0.000 | **239,479** | 239,479 |
| D4 | **205,832** | **205,832** | **205,832** | 205,832 | 0.000 | **205,832** | 205,832 |
| E1 | **238,880** | **238,880** | **238,880** | 238,880 | 0.000 | **238,880** | 238,880 |
| E2 | **212,263** | **212,263** | **212,263** | 212,263 | 0.000 | **212,263** | 212,263 |
| E3 | **206,659** | **206,659** | **206,659** | 206,659 | 0.000 | **206,659** | 206,659 |
| F1* | **263,173** | 267,060 | **263,174** | 263,173 | 0.000 | **263,173** | 263,173 |
| F2* | 265,493 | **265,214** | **265,214** | 265,214 | 0.000 | **265,213** | 265,213 |
| F3* | 241,120 | 241,970 | **241,121** | 241,120 | 0.000 | **241,120** | 241,120 |
| F4* | **233,861** | 235,175 | **233,862** | 233,861 | 0.000 | **233,861** | 233,861 |
| G1 | **306,306** | **306,305** | 306,537 | 306,305 | 0.000 | **306,306** | 306,305 |
| G2 | **245,441** | **245,441** | **245,441** | 245,441 | 0.000 | **245,441** | 245,441 |
| G3 | **229,507** | **229,507** | **229,507** | 229,507 | 0.000 | **229,507** | 229,507 |
| G4 | **232,521** | **232,521** | **232,521** | 232,521 | 0.000 | **232,521** | 232,521 |
| G5 | **221,730** | **221,730** | **221,730** | 221,730 | 0.000 | **221,730** | 221,730 |
| G6 | **213,457** | **213,457** | **213,457** | 213,457 | 0.000 | **213,457** | 213,457 |
| H1 | **268,933** | **268,933** | **268,933** | 268,933 | 0.000 | **268,933** | 268,933 |
| H2 | **253,365** | **253,366** | **253,366** | 253,365 | 0.000 | **253,365** | 253,365 |
| H3 | **247,449** | **247,449** | **247,449** | 247,449 | 0.000 | **247,449** | 247,449 |
| H4 | **250,221** | **250,221** | **250,221** | 250,221 | 0.000 | **250,221** | 250,221 |
| H5 | **246,121** | **246,121** | **246,121** | 246,121 | 0.000 | **246,121** | 246,121 |
| H6 | **249,135** | **249,135** | **249,135** | 249,135 | 0.000 | **249,135** | 249,135 |
| I1 | 350,435 | **350,245** | **350,245** | 350,245 | 0.000 | **350,246** | 350,245 |
| I2 | **309,944** | **309,944** | **309,944** | 309,944 | 0.000 | **309,944** | 309,944 |
| I3 | **294,507** | **294,507** | **294,507** | 294,507 | 0.000 | **294,507** | 294,507 |
| I4 | **295,988** | **295,988** | **295,988** | 295,988 | 0.000 | **295,988** | 295,988 |
| I5 | **301,236** | **301,236** | **301,236** | 301,236 | 0.000 | **301,236** | 301,236 |
| J1 | **335,007** | **335,007** | **335,007** | 335,007 | 0.000 | **335,006** | 335,006 |
| J2 | 310,793 | **310,417** | **310,417** | 310,417 | 0.000 | **310,417** | 310,417 |
| J3 | 279,306 | **279,219** | **279,219** | 279,219 | 0.000 | **279,219** | 279,219 |
| J4 | 296,860 | **296,533** | **296,533** | 296,533 | 0.000 | **296,533** | 296,533 |
| K1 | 394,974 | 394,376 | 395,076 | 394,376 | 0.077 | **394,071** | 394,071 |
| K2 | 363,829 | **362,130** | **362,130** | 362,130 | 0.000 | **362,130** | 362,130 |
| K3 | 366,246 | **365,694** | **365,694** | 365,694 | 0.000 | **365,694** | 365,694 |
| K4 | 351,345 | **348,949** | 349,870 | 348,949 | 0.000 | **348,950** | 348,949 |
| L1* | 426,401 | 426,013 | 417,922 | 417,922 | 0.006 | **417,896** | 417,896 |
| L2 | 402,152 | **401,229** | 401,248 | 401,229 | 0.000 | **401,228** | 401,228 |
| L3 | 404,391 | **402,678** | **402,678** | 402,678 | 0.000 | **402,678** | 402,678 |
| L4 | 384,999 | **384,636** | **384,636** | 384,636 | 0.000 | **384,636** | 384,636 |
| L5 | 389,044 | **387,565** | **387,565** | 387,565 | 0.000 | **387,565** | 387,565 |
| M1 | 400,384 | 398,914 | 398,730 | 398,730 | 0.034 | **398,593** | 398,593 |
| M2 | 398,924 | 399,336 | 397,324 | 397,324 | 0.102 | **396,917** | 396,917 |
| M3 | 377,433 | 377,212 | 377,329 | 377,212 | 0.402 | **375,696** | 375,696 |
| M4 | 349,091 | 348,418 | 348,418 | 348,418 | 0.080 | **348,140** | 348,140 |
| N1 | 409,531 | 410,789 | **408,101** | 408,101 | 0.000 | **408,101** | 408,101 |
| N2 | 408,287 | 409,385 | **408,065** | 408,065 | 0.000 | **408,066** | 408,065 |
| N3 | **394,338** | **394,338** | **394,338** | 394,338 | 0.000 | **394,338** | 394,338 |
| N4 | 399,029 | 398,965 | **394,788** | 394,788 | 0.000 | **394,788** | 394,788 |
| N5 | 376,522 | **373,476** | 373,723 | 373,476 | 0.000 | **373,476** | 373,476 |
| N6 | 374,774 | **373,759** | **373,759** | 373,759 | 0.000 | **373,759** | 373,759 |
| *average time(sec)* | *13.6* | *1.18* | *1.13* | | | *0.98* | |

**TS**: The *K-tree_r* Tabu Search version (Brandão, 2006) - Pentium III 500 MHz, C, **LNS**: The 6R-*no learning* version of the LNS metaheuristic (Ropke and Pisinger, 2006) - Pentium IV 1.5 GHz, C++, **MACS**: The metaheuristic proposed by Gajpal and Abad (2009) - Intel Xeon 2.4 GHz, C. **RPA**: The proposed metaheuristic - single core of Intel T5500 1.66 GHz, Visual C#. **BAS**: Best algorithmic solution among TS, LNS and MACS. **BKS**: The best known solution for each problem, ***%gap***$_{BAS}$**:** The percent gap between the BAS and the RPA solution scores (= 100 · (BAS - RPA) / BAS). * Ropke and Pisinger (2006) appear to have considered some different instance characteristics (see Table 4). Any slight discrepancies are due to different rounding schemes.

Table 4. Solution scores for five VRPB instances with different characteristics

| Instance | LNS | RPA | Instance | LNS | RPA |
|---|---|---|---|---|---|
| F1 | 267,060 | 267,060 | L1 | 426,013 | 426,014 |
| F2 | 265,214 | 265,213 | | | |
| F3 | 241,970 | 241,969 | | | |
| F4 | 235,175 | 235,175 | | | |

**LNS**: The 6R-*no learning* version of the LNS metaheuristic (Ropke and Pisinger, 2006), **RPA**: The proposed metaheuristic.
For the F group of instances, the demand of customer lying at (x, y) = (5103, 11065) is 1013. For the L1 instance the vehicle capacity is 4000 product units. Any slight discrepancies are due to different rounding schemes.

# FIGURES

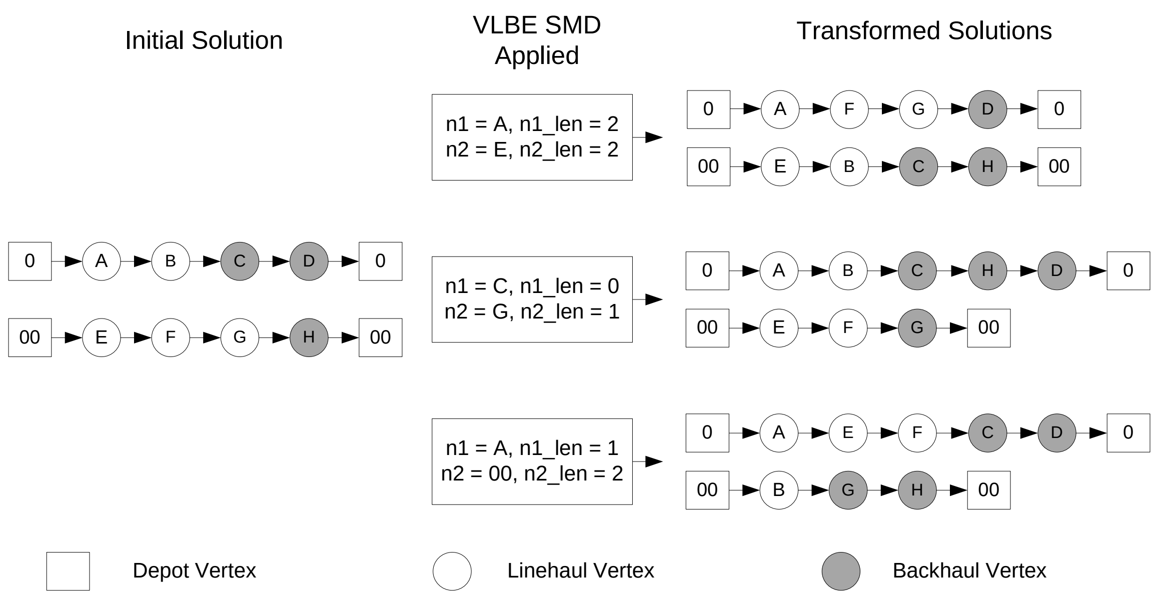


Fig 1. Applying a VLBE SMD to a VRPB solution

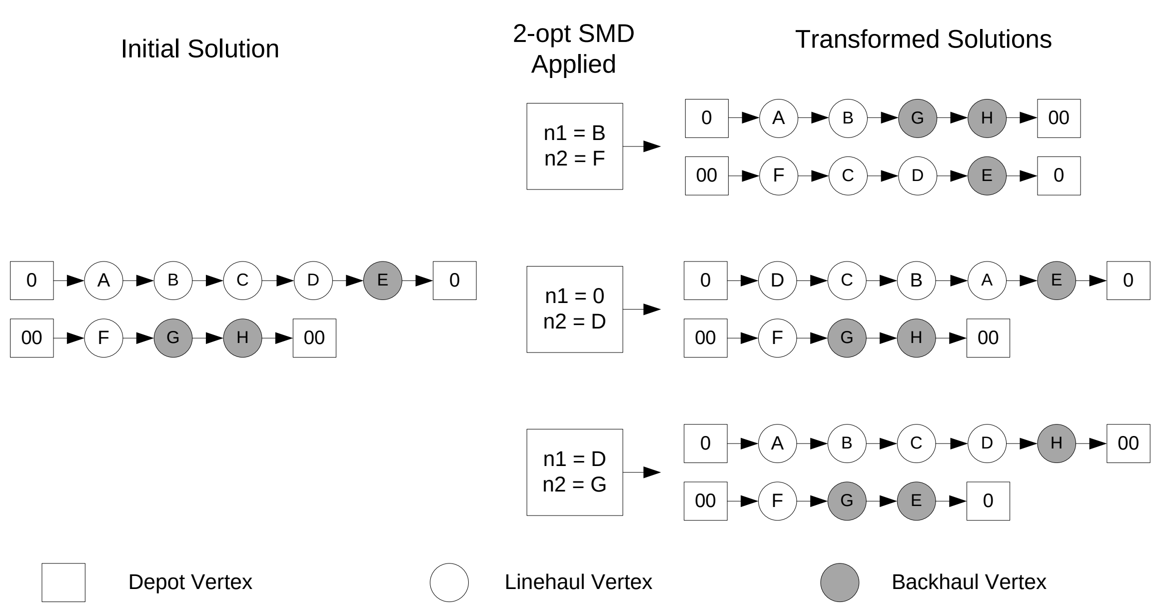


Fig 2. Applying a 2-opt SMD to a VRPB solution