Les Cahiers du GERAD ISSN: 0711–2440

A Tabu Search Algorithm for the Split Delivery Vehicle Routing Problem

C. Archetti, A. Hertz
M.G. Speranza

G–2003–18

March 2003

A Tabu Search Algorithm for the Split Delivery Vehicle Routing Problem

C. Archetti
Dipartimento di Metodi Quantitativi
Università degli Studi di Brescia, Italia
archetti@eco.unibs.it

A. Hertz
Département de Mathématiques et de Génie Industriel
École Polytechnique, Montréal, Canada
and GERAD
Alain.Hertz@gerad.ca

M.G. Speranza
Dipartimento di Metodi Quantitativi
Università degli Studi di Brescia, Italia
speranza@eco.unibs.it

March, 2003

Les Cahiers du GERAD
G–2003–18

Abstract

We describe a tabu search algorithm for the vehicle routing problem with split deliveries. At each iteration, a neighbour solution is obtained by removing a customer from a set of routes where it is currently visited, and by inserting it either into a new route, or into an existing route which has enough residual capacity. The algorithm also considers the possibility of inserting a customer into a route without removing it from another route. The insertion of a customer into a route is done by means of the cheapest insertion method. Computational experiments are reported for a set of benchmark problems, and the results are compared with those obtained by the algorithm proposed by Dror and Trudeau.

Résumé

Nous décrivons un algorithme tabou pour le problème de tournées de véhicules dans lequel les livraisons auprès de chaque client peuvent être réparties entre plusieurs véhicules. À chaque itération, une solution voisine est obtenue en supprimant un client d'un ensemble de routes le desservant, et en l'insérant dans une nouvelle route ou dans une route existante dont la capacité résiduelle est suffisante. L'algorithme considère également la possibilité d'insérer un client dans une route sans l'ôter d'une autre route. L'insertion d'un client dans une route est réalisée à l'aide de la méthode du plus petit détour. Nous présentons des expériences numériques sur des problèmes tests, et les résultats sont comparés avec ceux obtenus à l'aide d'un algorithme proposé par Dror et Trudeau.

1 Introduction

We consider the *Split Delivery Vehicle Routing Problem* (SDVRP) where a fleet of homogeneous vehicles has to serve a set of customers. Each customer can be visited more than once, contrary to what is usually assumed in the classical *Vehicle Routing Problem* (VRP) and the demand of each customer can be greater than the capacity of the vehicles. No constraint on the number of available vehicles is considered. There is a single depot for the vehicles and each vehicle has to start and end its tour at the depot. The objective is to find a set of vehicle routes that serve all the customers such that the sum of the quantities delivered in each tour does not exceed the capacity of the vehicles and the total distance travelled is minimized.

The SDVRP is a variant of the *Capacitated Vehicle Routing Problem* (CVRP) which is well known in the literature (for a survey of vehicle routing problems, see [14]). In [10] the authors have described a tabu search algorithm for the capacitated vehicle routing problem showing that this heuristic works well on this problem.

The SDVRP has been introduced in the literature only a few years ago. In [6] and [7] Dror and Trudeau have analyzed the savings generated by allowing split deliveries in a vehicle routing problem and they have presented a heuristic algorithm for the problem. They have shown that when the distances satisfy the triangle inequality there exists an optimal solution for the SDVRP where no pair of tours has two or more vertices in common. Valid inequalities for the SDVRP are described in [5] while real applications of the problem are studied in [12] and [13]. In [3] a lower bound is proposed for the SDVRP where the demand of each customer is lower than the capacity of the vehicles and the quantity delivered by the vehicles when visiting a customer is an integer number. In [8] the authors present a mathematical formulation and a heuristic algorithm for the SDVRP with grid network distances and time windows constraints. In [1] and [2] the authors have analyzed the k-SDVRP where each vehicle has a capacity equal to a given integer k, and where the demands of the customers as well as the quantity delivered by a vehicle when visiting a customer are integer numbers. They have proved that the problem is NP-hard when $k \geq 3$ and they have shown that, under specific conditions on the distances, the problem is reducible in polynomial time to a new problem where each customer has a demand that is lower than the capacity of the vehicles, with a possible reduction on the number of customers.

A *direct trip* in a k-SDVRP is a tour where a vehicle starts from the depot, goes directly to a customer where it delivers k units, and then turns back directly to the depot. Given an instance I of the k-SDVRP, one can build a *reduced instance*, denoted I_R, by modifying the demand d_i of each customer to $d_i - k\left\lfloor \frac{d_i}{k} \right\rfloor$. A solution s_R for I_R can then be transformed into a solution s for I by adding $\left\lfloor \frac{d_i}{k} \right\rfloor$ direct trips for each customer i. Now, given an instance I of the k-SDVRP, consider the algorithm that first determines an optimal solution s_R^* for the reduced instance I_R, and then builds the associated solution s^* for I. It is proved in [2] that this algorithm gives a worst case error of $\frac{3}{2}$ when the distances satisfy the triangle inequality.

In this paper we consider the k-SDVRP. We describe a tabu search algorithm to solve this problem that overcomes the typical problem of the tabu search algorithms: the tuning of the parameters. Actually, our algorithm is very simple and easy to implement, and there are only two parameters to be set: the length of the tabu list and the maximum number of iterations the algorithm can run without improvement of the best solution found.

The paper is organized as follows. In Section 2 we show that there always exists an optimal solution where the quantity delivered by each vehicle when visiting a customer is an integer number. In Section 3 we describe the tabu search algorithm for the k-SDVRP. In Section 4 Dror and Trudeau's algorithm [7] is described and computational results as well as comparisons with Dror and Trudeau's algorithm are reported in Section 5.

2 A mathematical formulation and some properties

The k-SDVRP can be defined over a graph $G = (V, E)$ with vertex set $V = \{0, 1, \ldots, n\}$ where 0 denotes the depot while the other vertices are the customers, and E is the edge set. The traversal cost (also called length) c_{ij} of an edge $(i, j) \in E$ is supposed to be non-negative. An integer demand d_i is associated with each customer $i \in V - \{0\}$. An unlimited number of vehicles is available, each with a capacity $k \in Z^+$. We will however consider an upper bound m on the number of vehicles needed to serve the customers. For example, one can use $m = \sum_{i=1}^{n} d_i$. Each vehicle must start and end its route at the depot. The demands of the customers must be satisfied, and the quantity delivered in each tour cannot exceed k. The objective is to minimize the total distance travelled by the vehicles. We give below a mixed integer programming formulation (P) for the k-SDVRP. We use the following notations:

x_{ij}^v is a boolean variable which is equal to 1 if vehicle v travels directly from i to j, and to 0 otherwise,

y_{iv} is the quantity of the demand of i delivered by the v-th vehicle,

M is a real positive number larger than or equal to $\sum_{i=1}^{n} d_i$.

The k-SDVRP can now be formulated as follows:

$$Min \sum_{i=0}^{n} \sum_{j=0}^{n} \sum_{v=1}^{m} c_{ij} x_{ij}^v \tag{1}$$

subject to:

$$\sum_{i=0}^{n} \sum_{v=1}^{m} x_{ij}^v \geq 1 \quad j = 0, \ldots, n \tag{2}$$

$$\sum_{i=0}^{n} x_{ip}^v - \sum_{j=0}^{n} x_{pj}^v = 0 \quad p = 0, \ldots, n;\ v = 1, \ldots, m \tag{3}$$

$$\sum_{i\in S}\sum_{j\in S} x_{ij}^v \leq |S|-1 \quad v=1,\ldots,m;\ S\subseteq V-\{0\} \tag{4}$$

$$y_{iv} \leq M\sum_{j=0}^{n} x_{ij}^v \quad i=1,\ldots,n;\ v=1,\ldots,m \tag{5}$$

$$\sum_{v=1}^{m} y_{iv} = d_i \quad i=1,\ldots,n \tag{6}$$

$$\sum_{i=1}^{n} y_{iv} \leq k \quad v=1,\ldots,m \tag{7}$$

$$x_{ij}^v \in \{0,1\} \quad i=0,\ldots,n;\ j=0,\ldots,n;\ v=1,\ldots,m \tag{8}$$

$$y_{iv} \geq 0 \quad i=1,\ldots,n;\ v=1,\ldots,m. \tag{9}$$

Constraints (2)-(4) are the classical routing constraints: constraints (2) impose that each vertex is visited at least once, (3) are the flow conservation constraints while (4) are the subtours elimination constraints. Constraints (5)-(7) concern the allocation of the demands of the customers among the vehicles: constraints (5) impose that customer i can be served by vehicle v only if v passes through i. Constraints (6) ensure that the entire demand of each vertex is satisfied while constraints (7) impose that the quantity delivered by each vehicle does not exceed the capacity.

We now show that there always exists an optimal integer solution to (P).

Theorem 1 *If (P) has feasible solutions, then there always exists an optimal solution in which variables $y_{iv} \in Z^+$.*

Proof. Let s^* denote an optimal solution of (P), let z^* be its value, and let $x_{ij}^{v^*}$ and y_{iv}^* be the corresponding optimal values of the variables. If all variables y_{iv}^* have an integer value, then there is nothing to prove. Otherwise, let A denote the set of pairs (i,v) with $i \in V-\{0\}$, $v \in \{1,\ldots,m\}$, and such that $\sum_{j=0}^{n} x_{ij}^{v^*} \geq 1$. For each customer i, define $N^+(i) = \{v|(i,v)\in A\}$. Similarly, define for each vehicle v the set $N^-(v) = \{i|(i,v)\in A\}$. Now, consider the following set of constraints:

$$\sum_{v\in N^+(i)} y_{iv} = d_i \quad i=1,\ldots,n \tag{10}$$

$$\sum_{i\in N^-(v)} y_{iv} \leq k \quad v=1,\ldots,m \tag{11}$$

$$y_{iv} \geq 0 \quad (i,v)\in A. \tag{12}$$

Constraints (10)-(12) are the classical ones of an assignment problem, and the y^*_{iv} values define a feasible assignment. Since the right-hand side of constraints (10)-(12) is made of integer numbers, it is well-known that there exists a feasible assignment a where each variable y_{iv} gets an integer value. By replacing the y^*_{iv} values in s^* by those in a, one gets a feasible solution to (P) that also has value z^* since variables y_{iv} do not appear in the objective function. ■

Dror and Trudeau [6] have shown another interesting property on optimal solutions of the SDVRP. To understand their result we first need to give the following definition.

Definition 1 *Consider a set $C = \{i_1, i_2, \ldots, i_t\}$ of customers and suppose that there exist t routes $r_1, \ldots, r_t$, $t \geq 2$, such that r_w contains customers i_w and i_{w+1}, $w = 1, \ldots, t-1$, and r_t contains customers i_1 and i_t. Such a configuration is called a t-split cycle.*

Dror and Trudeau have shown that, if the distances satisfy the triangle inequality, then there always exists an optimal solution to the SDVRP which does not contain t-split cycles, $t \geq 2$. In particular, for $t = 2$, this implies that there exists an optimal solution where each pair of tours has at most one vertex in common.

3 A tabu search algorithm for the k-SDVRP

In this section we present a tabu search algorithm for the k-SDVRP, called SPLITABU. It is a very simple algorithm, easy to implement, where there are only two parameters to be set: the length of the tabu list and the maximum number of iterations the algorithm can run without improvement of the best solution found. The algorithm is composed of the three following phases.

- *Phase 1:* construction of an initial feasible solution.
- *Phase 2:* tabu search phase.
- *Phase 3:* final improvement of the solution found by the tabu search phase.

3.1 Phase 1: construction of an initial feasible solution

For constructing an initial feasible solution, we first create a reduced instance by creating as many direct trips as possible (see Section 1). We then solve a traveling salesman problem on the reduced instance and we cut this giant tour into pieces so that the capacity constraints are satisfied. For building a giant TSP tour, we use the GENIUS algorithm proposed by Gendreau, Hertz and Laporte [9]. GENIUS is composed of two procedures: the first one, GENI, is a generalized insertion procedure and the second one, US, is a postoptimization routine. It is shown in [9] that this algorithm is a very efficient solution method for the traveling salesman problem. A detailed description of Phase 1 is given below.

Phase 1

1. Create $\left\lfloor \frac{d_i}{k} \right\rfloor$ direct trips for each customer i, and consider the reduced instance I_R where each customer i has a demand equal to $d_i - k \left\lfloor \frac{d_i}{k} \right\rfloor$.
   Remove all customers having no demand in I_R .
2. Construct a TSP tour T for the reduced instance I_R by means of the GENIUS algorithm.
3. Choose an orientation for T and label the customers so that $T = (0 = v_0, v_1, \ldots, v_{b-1}, v_b = 0)$.
4. If the total demand on T is smaller than or equal to k then STOP.
   Determine the smallest index i such that the demand on T up to v_i is strictly larger than k.
   Build a route $(0, v_1, \ldots, v_{i-1}, 0)$, set $T := (0, v_i, \ldots, v_{b-1}, 0)$ and go to 3.

3.2 Phase 2: tabu search phase

The tabu search algorithm uses two procedures called ORDER ROUTES and BEST NEIGHBOUR. Each procedure has a customer i as input parameter. In what follows we denote by d_{ir} the quantity delivered to i on route r, by ρ_r the residual capacity on route r (i.e. $\rho_r = k - \sum_{i=1}^{n} d_{ir}$), and by $f(s)$ the total travelled distance in solution s. Given a customer i, the first procedure constructs an ordered list O_i of the routes visiting i, the routes being ordered according to the saving obtained by removing i. This is performed as follows.

ORDER ROUTES(*i*)

1. Determine the set U_i of routes that visit customer i.
2. For each route u in U_i compute $s_u = c_{pi} + c_{iq} - c_{pq}$ where p and q are the predecessor and the successor of i on u, respectively.
3. Order the routes in U_i by non-increasing value of s_u and set O_i equal to this ordered list.

In our tabu search algorithm, a move from a solution s to a neighbour solution s' is performed by including a customer i into a route r and by removing i from a subset $U \subseteq O_i - \{r\}$ of routes visiting i. The subset U is determined on the basis of the ordered list established by ORDER ROUTES(i). When a customer i is added to a route r, we consider as tabu for θ iterations the removal of i from r, and we will say that route r is tabu for i. Also, when a customer i is removed from a route u, then it is tabu for θ iterations to reinsert i into u, and we will also say that u is tabu for i. We have observed that values of θ that depend on the number n of customers and on the number g of routes in the current solution s produce better solutions. According to preliminary experiments,

we have decided to set θ equal a random integer number in the interval $[\sqrt{10}, \sqrt{10+p}]$ where $p = n + g$ if $n + g < 100$ and $p = \frac{3}{2}(n + g)$ if $n + g \geq 100$.

The tabu restrictions defined above may be too strong and forbid a good neighbour solution. For this reason, we also consider the possibility of removing a customer i from routes that are tabu for i and to insert i into a route r which is tabu for i. However, such a neighbour is only accepted if it leads to a better solution than the best one encountered so far.

Given a customer i to be inserted in a route r, it may happen that each route u in $O_i - \{r\}$ with $d_{iu} \leq \rho_r$ is tabu for i. In such a case, we consider the first non-tabu route u for i in $O_i - \{r\}$, we decrease by ρ_r units the quantity delivered to i in u, and we insert customer i in r, setting $d_{ir} = d_{ir} + \rho_r$. Hence, part of the demand of i is removed from route u, and customer i is inserted into r. This cannot lead to a solution of better value than s and this is the reason why we do not allow such a move if u or r is tabu for i.

Given a customer i, procedure BEST NEIGHBOUR(i) described below determines a candidate neighbour s_i. In phase 2, this procedure is called for each customer in $V - \{0\}$ and a move is performed from the current solution s to the best neighbour among $s_1, \ldots, s_n$. In what follows we denote by R the set containing all routes plus a new route (i.e., a route that does not visit any customer) and $U_i^T = \{u \in O_i | u \text{ is tabu for } i\}$.

***BEST NEIGHBOUR**(i)*

Input: a solution s and the best solution s^* encountered so far.

Output: a neighbour solution s_i, a route r^* and a subset of routes $U^* \subseteq O_i - \{r^*\}$; solution s_i is obtained from s by inserting i into r^* and by either setting $d_{ir^*} := d_{ir^*} + \rho_{r^*}$ and $d_{iu} := d_{iu} - \rho_{r^*}$ if U^* contains a single route u with $d_{iu} > \rho_{r^*}$, or else by removing i from all routes in U^*.

1. Set $BestValue := \infty$ and construct the ordered list O_i by means of ORDER ROUTES(i).
2. For each route r in R do the following

   2.1 *Removal from non-tabu routes and insertion into a non-tabu route*

   1. If r is not tabu for i then set $\rho := \rho_r$ and $U := \emptyset$; else go to 2.2.
   2. Consider all routes $u \in O_i - \{r\}$ according to the order defined in O_i and do the following: if u is not tabu for i and $d_{iu} \leq \rho$ then set $U := U \cup \{u\}$ and $\rho := \rho - d_{iu}$.
   3. If $U = \emptyset$ then set $F := \infty$; else let s' be the solution obtained from s by removing i from all routes in U and by inserting i into r setting $d_{ir} := d_{ir} + \sum_{u \in U} d_{iu}$, and let $F = f(s')$.
   4. If $U = \emptyset$, let u be the first non-tabu route for i in $O_i - \{r\}$: set $U := \{u\}$ and consider the solution s' obtained by inserting i into r and by setting $d_{iu} := d_{iu} - \rho$ and $d_{ir} := d_{ir} + \rho$, and let $F := f(s')$.

5. If $F < BestValue$ then set $U^* := U$, $r^* := r$, $BestValue := F$ and $s_i := s'$.

2.2 *Removal from tabu routes or/and insertion into a tabu route*

1. If r is tabu for i or U_i^T contains a route u with $d_{iu} \leq \rho_r$ then set $\rho := \rho_r$, otherwise return to 2.1 with the next route $r \in R$. If r is tabu for i then set $U := \emptyset$, else determine the first route u in U_i^T (according to the ordering in $O(i)$) such that $d_{iu} \leq \rho_r$, and set $U := \{u\}$ and $\rho := \rho_r - d_{iu}$.
2. Consider all routes $u \in O_i - (U \cup \{r\})$ according to the order defined in O_i and do the following: if $d_{iu} \leq \rho$ then set $U := U \cup \{u\}$ and $\rho := \rho - d_{iu}$.
3. If $U = \emptyset$ then set $F := \infty$; else let s' be the solution obtained from s by removing i from all routes in U and by inserting i into r setting $d_{ir} := d_{ir} + \sum_{u \in U} d_{iu}$, and let $F = f(s')$.
4. If $F < BestValue$ and $F < f(s^*)$ then set $U^* := U$, $r^* := r$, $BestValue := F$ and $s_i := s'$.
5. Return to 2.1 with the next route $r \in R$.

We can now describe the tabu search phase. It is a standard tabu search algorithm that stops when n_{max} iterations have been performed without improvement of the best solution encountered so far. According to preliminary experiments we have fixed $n_{\max}$ equal to $400n$.

Phase 2

1. Let s be the solution produced by Phase 1. Set $s^* := s$ and $counter := 0$;
2. Set $Best := \infty$
   For i =1 to n do
   Call BEST NEIGHBOUR(i)
   If $f(s_i) < Best$ then set $BestI := i$, $BestU := U^*$, $Bestr := r^*$,
   $BestS := s_i$ and $BestF := f(s_i)$;
3. Set $s := BestS$;
   Consider $Bestr$ as well as all routes in $BestU$ as tabu for $BestI$ during θ iterations.
   If $BestF < f(s^*)$ then set $s^* := s$ and $counter := 0$; else set $counter := counter + 1$;
   If $counter < n_{max}$ then go to 2; else STOP.

3.3 Phase 3: final improvement

As mentioned in Section 2, Dror and Trudeau have shown that, if the distances satisfy the triangle inequality, then there always exists an optimal solution to the SDVRP which does not contain t-split cycles with $t \geq 2$. A t-split cycle can easily be removed from a solution as follows. Suppose that there exists a t-split cycle with routes $r_1, \ldots, r_t$ and such that r_w contains customers i_w and i_{w+1}, $w = 1, \ldots, t-1$, and r_t contains customers i_1 and i_t. Let w^* be an index such that $d_{i_{w^*} r_{w^*}} \leq d_{i_w r_w}$, $w = 1, \ldots, t$: one can transfer $d_{i_{w^*} r_{w^*}}$ units of

demand of each customer i_w, $w = 1, \ldots, t-1$, from r_w to r_{w+1} as well as the same quantity for customer i_t from r_t to r_1. Customer i_{w^*} will thus be removed from route r_{w^*}. If the distances satisfy the triangle inequality, then this new solution is possibly better than the one with the t-split cycle.

The final improvement phase performs such kind of improvements. It also tries to reduce the length of each route by applying the GENIUS algorithm.

Phase 3

1. Let s be the solution resulting from Phase 2. If the distances satisfy the triangle inequality, then delete all t-split cycles from s (if any).
2. Improve each individual route of s by means of the GENIUS algorithm.

4 Dror and Trudeau's algorithm

We give in this section a short description of the algorithm proposed by Dror and Trudeau for the SDVRP [6]. They consider the case where the demand of each customer is lower than the capacity of the vehicles. Their heuristic is a local search algorithm and is composed of the following two main procedures.

K-SPLIT INTERCHANGE

Consider a vertex i and its total demand d_i :

1. Remove vertex i from all the routes where it is visited.
2. Consider all subsets R of routes such that the total residual capacity $\sum_{r \in R} \rho_r$ is greater than or equal to d_i. For each such subset R compute the total insertion cost of i into all routes of R. Choose the subset R that leads to the least insertion cost and insert i into all routes of R.

ROUTE ADDITION

Consider a customer i which appears in at least two routes r_1 and r_2. Eliminate the split of i on these two routes and create a new route in the following way:

1. Preserve the four principle route segments on r_1 and r_2 (from the depot to the vertex preceding i and from the vertex succeeding i to the depot).
2. Create three routes considering all the possible combinations between the principle route segments and i (which must not be split) and choose the best one.

There are 9 possible combinations (for details see [6]). The same procedure is considered when customer i is split among 3 different routes. In this case there are 19 possible combinations to be considered. If a vertex is visited by more than 3 routes, the algorithm considers all the possible combinations of 2 and 3 routes.

Moreover, Dror and Trudeau use the following classical improvement procedures which have been developed for the capacitated VRP.

NODE INTERCHANGES This procedure is based on one-node moves and two-nodes swaps between routes and is described in detail in [4].

2-OPT This is the classical 2-opt procedure for the TSP [11].

Defining boolean variables *split_impr* and *add_impr*, the main algorithm works as follows:

Dror and Trudeau's algorithm

1. Construct a feasible VRP solution.
2. NODE INTERCHANGES: execute all node interchange improvements.
3. 2-OPT: execute all 2-opt route improvements.
4. Set *split_impr* = "false" and *add_impr* = "false".
5. K-SPLIT INTERCHANGE: execute all k-split interchange improvements. If there is at least one improvement then set *split_impr* = "true".
6. ROUTE ADDITION: execute all route addition improvements. If there is at least one improvement then set *add_impr* = "true".
7. If *add_impr* = "true" then go to step 5. Otherwise, if *split_impr* = "true" go to step 2 else STOP.

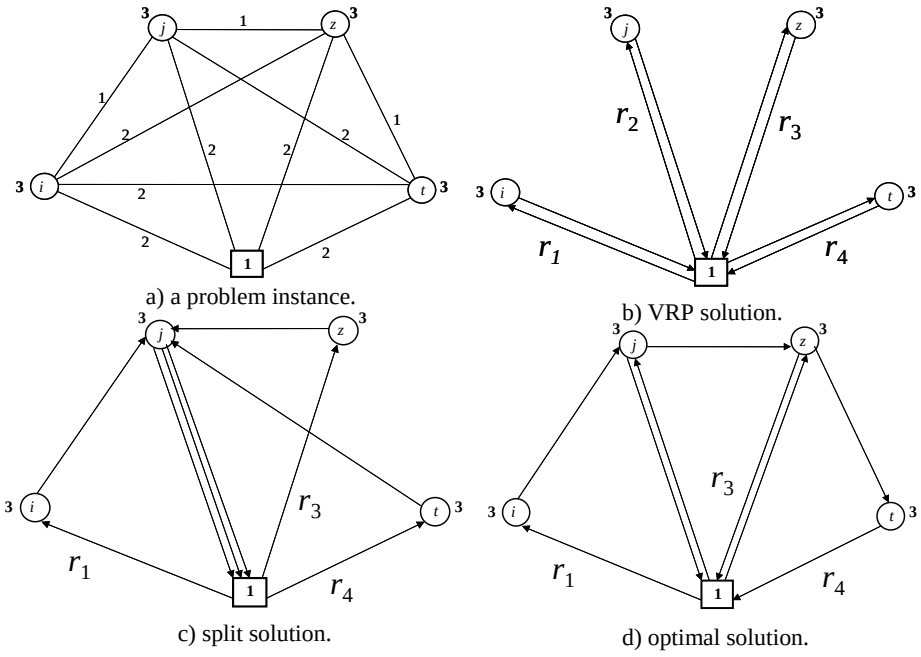

Figure 1: An instance of the 4-SDVRP for which Dror and Trudeaus algorithm cannot find the optimal solution.

We close this section by showing that there exist instances for which the above algorithm has no chance to find an optimal solution. Consider the instance of Figure 1a which is a 4-SDVRP with symmetrical distances that satisfy the triangle inequality. Each of the four customers has a demand equal to three. In Figure 1b we have represented the unique feasible VRP solution (where no split is allowed) which is therefore the initial solution of Dror and Trudeau's algorithm (and also of SPLITABU). This solution has a cost of 16. Procedures NODE INTERCHANGES and 2-OPT cannot improve this solution. As the demand of each customer is three while the residual capacity of each route is one, the only feasible k-split interchange consists in splitting the demand of one customer among the three other routes as shown in Figure 1c (where the demand of customer j is split into routes r_1, r_3 and r_4). Such a k-split interchange produces a solution of value 16 (if the demand of customers j or z is split) or 17 (if the demand of customers i or t is split). Hence, no improvement can be obtained with procedure K-SPLIT INTERCHANGE and one therefore enters step 6 with the initial VRP solution. Since there is no split vertex, the ROUTE ADDITION procedure cannot perform any change on this solution, and Dror and Trudeau's algorithm therefore stops at step 7 without finding the optimal solution of value 15 which is represented in Figure 1d (where r_1 delivers three units to i and one unit to j, route r_3 delivers two units to j and to z and route r_4 delivers one unit to z and three units to t).

With the neighbourhood defined in SPLITABU, it is possible to reach the optimal solution. Indeed, one can first move one unit of demand of customer j from route r_2 to route r_1 (Figure 2a). One can then move one unit of demand of customer z from route r_3 to route r_4 (Figure 2b). One can finally remove customer j from route r_2 and insert it into r_3 with $d_{j3} = 2$ (Figure 1d). It is in fact easy to build such instances for any value of k (not only for $k = 4$).

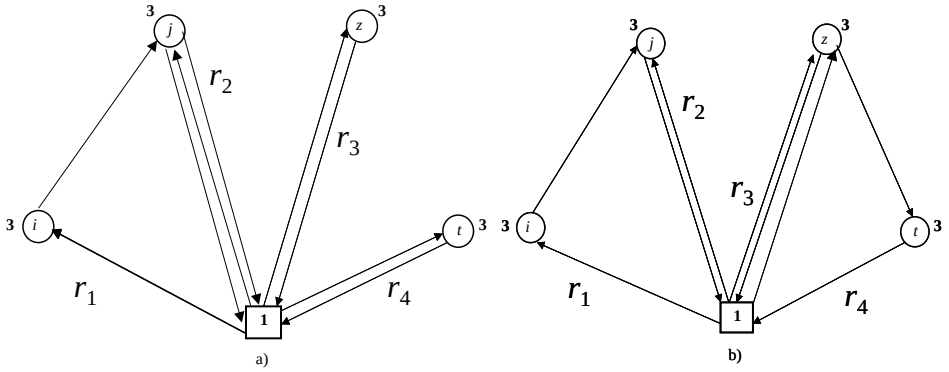

Figure 2: Solutions visited by SPLITABU for the same instance as in Figure 1.

5 Computational results

This section contains computational experiments to evaluate the performance of SPLITABU. The algorithm was implemented in C++ on a PC Pentium 4, 256 MB RAM. We have first generated problems of small size as follows. We have considered the n first customers of the benchmark VRP problems described in [10], with $n = 6, 7, \ldots, 15$. All these problems have a fixed vehicle capacity which was reduced to a smaller value so that at least three vehicles are needed in a VRP solution. These small problems were solved using CPLEX 6.6. While problems with up to 10 customers could be solved in a few seconds, larger problems with $n = 11, \ldots, 15$ required between one hour and four days of computation. For comparison, SPLITABU has produced the same optimal solutions for all these problems in less than one second.

Larger problems, with more than 15 customers can hardly be solved to optimality. In order to evaluate the performance of SPLITABU on such larger instances we compare the results produced by SPLITABU with those obtained using Dror and Trudeau's algorithm (DT for short). Algorithm DT was implemented in C++ on the same PC. We have considered problems 1-5, 11 and 12 from [10]. These problems have between 50 and 199 customers. As proposed by Dror and Trudeau [6], the demands of the customers have been modified as follows. Let α and γ be two parameters chosen in the interval $[0, 1]$, with $\alpha \leq \gamma$. The demand d_i of customer i is set equal to

$$d_i = \lfloor \alpha k + \delta (\gamma - \alpha) k \rfloor$$

where δ is a random number in $[0, 1]$. In words, the demand d_i of customer i is chosen randomly in the interval $[\alpha k, \gamma k]$. As in [6], we have considered the following combinations (α, γ) of parameters: $(0.01, 0.1)$, $(0.1, 0.3)$, $(0.1, 0.5)$, $(0.1, 0.9)$, $(0.3, 0.7)$ and $(0.7, 0.9)$. We have also considered the case where the original demands are not changed. This gives a total of 49 instances (since there are seven instances with different demands for each of the seven VRP problems taken from [10]).

Preliminary experiments with SPLITABU have shown that some solutions can easily be improved by applying the 2-OPT and the NODE INTERCHANGES procedures each time the best solution s^* encountered so far is improved (step 3 of Phase 2). This variant of SPLITABU is called SPLITABU-DT. We have also implemented another variant of SPLITABU where the GENIUS algorithm [9] is applied on each route of s^* each time s^* is improved. The results for this last variant are not reported here since SPLITABU-DT has always produced solutions that are better both in terms of quality and CPU time. Finally, since algorithm DT is much faster than our tabu search (see Table 1), we have considered a variant of SPLITABU-DT, called FAST-SPLITABU where Phase 2 is run for at most one minute.

Computational results are given in Tables 1 and 2. Each variant of SPLITABU was run 5 times on each instance (two executions on a same instance may differ due to the randomness of the length of the tabu list). Computing times are reported in Table 1 where

				D & T	SPLITABU			SPLITABU-DT		
Problem	*n*	**α**	**γ**	**CPU**	**CPU_{min}**	**CPU_{av}**	**CPU_{max}**	**CPU_{min}**	**CPU_{av}**	**CPU_{max}**
1	50	-	-	0	13	17	22	7	13	24
2	75	-	-	0	23	64	175	21	36	70
3	100	-	-	0	31	60	93	34	58	120
4	150	-	-	0	193	440	809	135	389	573
5	199	-	-	0	665	1900	4170	281	386	465
6	120	-	-	1	39	40	41	38	38	39
7	100	-	-	0	65	86	105	41	49	68
1	50	0.01	0.1	0	5	9	14	4	5	7
2	75	0.01	0.1	0	17	42	86	10	13	16
3	100	0.01	0.1	0	27	59	89	23	31	45
4	150	0.01	0.1	0	107	258	810	107	173	233
5	199	0.01	0.1	0	224	754	1513	319	526	858
6	120	0.01	0.1	1	39	61	102	37	42	52
7	100	0.01	0.1	0	23	71	118	23	58	92
1	50	0.1	0.3	0	9	27	44	12	22	33
2	75	0.1	0.3	0	27	78	180	25	45	71
3	100	0.1	0.3	0	60	122	193	66	96	121
4	150	0.1	0.3	0	221	545	900	265	393	596
5	199	0.1	0.3	1	660	1224	2399	630	755	1061
6	120	0.1	0.3	1	214	516	1117	110	143	198
7	100	0.1	0.3	0	67	85	119	67	146	440
1	50	0.1	0.5	0	21	56	141	13	28	62
2	75	0.1	0.5	0	38	71	89	70	123	204
3	100	0.1	0.5	0	92	206	562	97	136	271
4	150	0.1	0.5	0	381	564	730	397	739	1512
5	199	0.1	0.5	1	1023	3811	7994	882	2668	4767
6	120	0.1	0.5	1	206	259	428	177	268	465
7	100	0.1	0.5	0	94	188	478	87	293	924
1	50	0.1	0.9	0	22	34	59	19	61	97
2	75	0.1	0.9	0	99	311	627	134	193	354
3	100	0.1	0.9	0	197	412	980	194	649	963
4	150	0.1	0.9	0	696	1822	2570	725	2278	4760
5	199	0.1	0.9	2	1883	2598	3162	2066	3297	4204
6	120	0.1	0.9	1	389	1037	2514	316	878	2739
7	100	0.1	0.9	0	187	523	1098	174	260	413
1	50	0.3	0.7	0	18	52	97	28	49	99
2	75	0.3	0.7	0	120	184	301	74	129	219
3	100	0.3	0.7	0	182	454	730	174	810	1638
4	150	0.3	0.7	1	800	1512	2535	839	3008	5533
5	199	0.3	0.7	2	1886	2279	3573	2067	3566	6799
6	120	0.3	0.7	1	368	477	830	451	659	1264
7	100	0.3	0.7	0	188	411	748	350	778	1321
1	50	0.7	0.9	0	61	160	267	83	106	144
2	75	0.7	0.9	0	210	437	625	582	869	1530
3	100	0.7	0.9	0	1497	1891	2160	762	1398	2686
4	150	0.7	0.9	2	4329	8783	17565	4837	10223	14005
5	199	0.7	0.9	7	4494	11347	31516	10285	21849	39401
6	120	0.7	0.9	1	1150	2033	2909	1189	1826	2674
7	100	0.7	0.9	0	928	1865	3074	506	1004	1426

Table 1: Computational times

				SPLITABU			SPLITABU-DT			FAST-SPLIT		
Problem	n	α	γ	z_{min}	z_{av}	z_{max}	z_{min}	z_{av}	z_{max}	z_{min}	z_{av}	z_{max}
1	50	-	-	**10.06**	9.65	9.11	9.55	9.06	7.99	9.55	9.06	7.99
2	75	-	-	**6.08**	4.82	3.95	5.48	5.06	4.53	5.48	5.06	4.53
3	100	-	-	7.55	6.21	4.48	**7.63**	7.38	6.75	**7.63**	7.37	6.75
4	150	-	-	**6.52**	6.06	5.12	5.87	5.38	4.79	4.91	3.75	2.29
5	199	-	-	1.34	0.57	0.16	**2.60**	2.39	2.04	2.46	2.13	1.58
6	120	-	-	0.73	-0.05	-0.64	**2.83**	2.60	2.37	**2.83**	2.60	2.37
7	100	-	-	**13.89**	13.57	13.44	**13.89**	13.28	12.34	**13.89**	13.28	12.34
1	50	0.01	0.1	**5.75**	5.12	4.70	**5.75**	5.14	4.65	**5.75**	5.14	4.65
2	75	0.01	0.1	8.31	7.77	7.45	**8.68**	8.14	7.62	**8.68**	8.14	7.62
3	100	0.01	0.1	6.20	4.27	1.66	**8.65**	6.82	5.11	**8.65**	6.82	5.11
4	150	0.01	0.1	7.34	6.45	5.58	**7.81**	6.87	6.02	7.47	6.59	6.02
5	199	0.01	0.1	**8.80**	6.32	3.85	8.57	7.84	7.26	7.55	6.75	5.51
6	120	0.01	0.1	**3.59**	0.73	-1.52	2.51	1.73	0.98	2.51	1.73	0.98
7	100	0.01	0.1	16.71	13.84	10.05	**17.34**	15.67	13.44	16.46	15.02	13.44
1	50	0.1	0.3	**6.72**	5.02	2.62	6.61	5.40	3.76	6.61	5.40	3.76
2	75	0.1	0.3	**5.28**	3.92	2.68	4.89	4.25	3.46	4.89	4.25	3.46
3	100	0.1	0.3	**1.44**	0.75	0.10	1.41	1.03	0.66	1.22	0.94	0.66
4	150	0.1	0.3	1.09	0.46	-0.17	**2.14**	1.61	1.17	1.62	1.28	0.86
5	199	0.1	0.3	1.01	0.68	0.48	**2.40**	2.15	1.94	1.99	1.76	1.48
6	120	0.1	0.3	**3.19**	1.53	0.86	1.07	0.92	0.62	0.99	0.84	0.62
7	100	0.1	0.3	9.02	7.91	6.73	**9.97**	8.47	6.35	9.95	8.14	6.35
1	50	0.1	0.5	**5.35**	4.05	2.62	5.05	3.96	3.34	5.05	3.96	3.34
2	75	0.1	0.5	3.32	2.69	1.85	**3.89**	3.13	2.73	3.44	2.72	2.02
3	100	0.1	0.5	**1.78**	1.31	1.06	1.42	0.95	0.74	1.42	0.82	0.11
4	150	0.1	0.5	2.20	1.56	0.76	**2.50**	1.61	1.17	1.50	0.94	0.39
5	199	0.1	0.5	1.44	0.78	0.14	**2.32**	1.20	-0.28	1.51	0.31	-0.28
6	120	0.1	0.5	**4.72**	3.72	2.56	4.54	2.81	0.17	2.48	1.52	0.17
7	100	0.1	0.5	7.13	5.39	4.89	**7.65**	5.39	3.18	6.15	4.27	3.10
1	50	0.1	0.9	4.12	3.32	2.20	**6.56**	4.87	2.83	**6.56**	4.84	2.83
2	75	0.1	0.9	2.86	2.49	2.03	**3.13**	2.34	1.78	1.95	1.52	1.04
3	100	0.1	0.9	2.79	2.27	1.97	**5.03**	3.39	2.14	2.93	1.88	0.01
4	150	0.1	0.9	1.72	1.03	0.22	**4.90**	3.42	-0.20	3.90	1.49	-0.20
5	199	0.1	0.9	1.90	1.24	0.62	**7.16**	4.88	3.94	2.23	1.14	-0.49
6	120	0.1	0.9	0.37	-0.44	-1.66	**4.16**	-0.81	-3.71	-2.27	-3.05	-4.11
7	100	0.1	0.9	3.63	2.43	0.57	**6.99**	4.18	1.34	6.45	3.94	1.34
1	50	0.3	0.7	3.69	2.72	1.37	**4.15**	3.52	2.83	**4.15**	3.51	2.85
2	75	0.3	0.7	1.76	1.34	0.90	**2.53**	2.04	1.60	2.23	1.87	1.60
3	100	0.3	0.7	3.49	2.85	2.11	**7.87**	4.33	1.84	3.61	2.61	1.84
4	150	0.3	0.7	2.40	1.83	0.87	**6.09**	4.37	2.98	3.02	2.02	1.40
5	199	0.3	0.7	-0.39	-1.07	-2.36	**5.85**	3.94	2.41	1.72	-0.01	-2.77
6	120	0.3	0.7	0.99	-1.16	-4.88	**5.23**	2.18	-0.68	2.67	-0.61	-2.16
7	100	0.3	0.7	4.11	2.33	1.19	**12.63**	7.91	4.38	4.85	3.36	1.47
1	50	0.7	0.9	0.76	0.63	0.34	**1.90**	1.14	0.70	**1.90**	1.01	0.64
2	75	0.7	0.9	-0.49	-0.83	-1.39	**2.02**	0.69	-0.41	-0.76	-1.22	-1.72
3	100	0.7	0.9	-0.65	-0.81	-0.95	**1.27**	0.72	0.29	-1.20	-1.96	-2.88
4	150	0.7	0.9	-0.63	-0.94	-1.68	**3.10**	1.56	0.88	-2.38	-3.47	-4.19
5	199	0.7	0.9	-1.28	-2.97	-6.24	**5.28**	1.96	0.47	-4.78	-5.96	-7.70
6	120	0.7	0.9	-1.73	-4.13	-5.85	**0.58**	-1.70	-3.58	-8.60	-8.64	-10.37
7	100	0.7	0.9	0.95	0.40	0.09	**4.16**	2.27	1.19	1.38	0.30	-0.77
AVERAGE				**3.81**	**2.81**	**1.72**	**5.38**	**4.03**	**2.81**	**3.72**	**2.76**	**1.73**

Table 2: Percentage improvement over Dror and Trudeaus algorithm

CPU_{av}, $CPU_{\min}$ and $CPU_{\max}$ represent the average, the minimum and the maximum CPU times in seconds calculated over the five tests made for each of the 49 instances. When no value is given for parameters α and γ, this means that we used the original demands taken from [10]. We observe that algorithm DT is very fast since it requires less than one second for 35 of the 49 instances. In average, SPLITABU and SPLITABU-DT require less than 10 minutes for 35 and 31 instances, respectively. They both require more than one hour in only two cases. We can observe in Table 1 that CPU times increase not only with the number of customers but also with their demands. Indeed, an instance with (α, γ)=$(0.01, 0.1)$ is typically solved much faster than the same instance with (α, γ)=$(0.7, 0.9)$. The reason is that more vehicles are needed when the demands are becoming larger, and this induces an increase in the number of neighbour solutions to be considered at each step of the tabu search.

Table 2 indicates the improvement in quality obtained by using the variants of SPLITABU instead of algorithm DT. For each algorithm, we give the average z_{av}, the smallest $z_{\min}$ and the largest $z_{\max}$ improvements (in %) over DT. The best results are indicated with bold numbers. It can be observed that SPLITABU-DT finds better solutions than DT on all 49 instances. The improvement even reaches 17.34% on instance 7 with (α, γ)=$(0.01, 0.1)$. Algorithms SPLITABU and FAST-SPLITABU both outperform DT on 43 instances. The advantage of the three variants of SPLITABU over algorithm DT is particularly visible on instances with small demands. We have observed in Table 1 that high demands induce large neighbourhoods for tabu search since more vehicles are needed. Hence, the optimization process in SPLITABU is more time consuming for such kind of instances, and this explains why FAST-SPLITABU (that stops after one minute) produces results of relatively bad quality on most instances with (α, γ)=$(0.7, 0.9)$. Notice however that SPLITABU-DT is able to gain up to 5 percents on these instances. The last line of Table 2 contains the average results over the 49 instances. It can clearly be observed that all tabu search algorithms improve on algorithm DT, even in the worst case (column $z_{\max}$).

Conclusions

We have described in this paper several variants of a tabu search algorithm, called SPLITABU, for the k-SDVRP. SPLITABU is a very simple algorithm, easy to implement, with only two parameters to be set. Computational experiments confirm that optimal solutions can be obtained in a practically null time for small instances having up to 15 customers. Comparison with Dror and Trudeau's algorithm (the only existing heuristic algorithm for the k-SDVRP) shows that the variants of SPLITABU provide almost always better solutions even when computational times are limited to one minute. Finally, we have shown that one can build instances for which Dror and Trudeau's algorithm cannot find the optimal solution, while the neighbourhood defined in SPLITABU overcomes this difficulty.

Acknowledgments

We wish to thank Jean-François Cordeau and Oli Madsen for providing the code for GENIUS and Moshe Dror for useful information on the code of his algorithm for the SDVRP.

References

[1] C. Archetti, R. Mansini, M.G. Speranza, "Complexity and reducibility of the skip delivery problem", *Transportation Science* (to appear).

[2] C. Archetti, M.G. Speranza, "A direct trip algorithm for the k-split delivery vehicle routing problem", Technical report n. 205, Department of Quantitative Methods, University of Brescia (2002) (submitted).

[3] J.M. Belenguer, M.C. Martinez, E. Mota, "A lower bound for the split delivery vehicle routing problem", *Operations Research* 48, 801–810 (2000).

[4] M. Dror, L. Levy, "A vehicle routing improvement algorithm comparison of a "greedy" and a matching implementation for inventory routing", *Computers and Operations Research* 13, 33–45 (1986).

[5] M. Dror, G. Laporte, P. Trudeau, "Vehicle routing with split deliveries", *Discrete Applied Mathematics* 50, 239–254 (1994).

[6] M. Dror, P. Trudeau, "Savings by split delivery routing", *Transportation Science* 23, 141–145 (1989).

[7] M. Dror, P. Trudeau, "Split delivery routing", *Naval Research Logistics* 37, 383–402 (1990).

[8] P.W. Frizzell, J.W. Giffin, "The split delivery vehicle scheduling problem with time windows and grid network distances", *Computers and Operations Research* 22, 655–667, 1995.

[9] M. Gendreau, A. Hertz, G. Laporte, "New insertion and postoptimization procedures for the traveling salesman problem", *Operations Research* 40, 1086–1094 (1992).

[10] M. Gendreau, A. Hertz, G. Laporte, "A tabu search heuristic for the vehicle routing problem", *Management Science* 40, 1276–1290 (1994).

[11] S. Lin, "Computer solutions of the traveling salesman problem", *Bell System Technical Journal* 44, 2245–2269 (1965).

[12] P.A. Mullaseril, M. Dror, J. Leung, "Split-delivery routing in livestock feed distribution", *Journal of the Operational Research Society* 48, 107–116 (1997).

[13] G. Sierksma, G.A. Tijssen, "Routing helicopters for crew exchanges on off-shore locations", *Annals of Operations Research* 76, 261–286 (1998).

[14] P. Toth, D. Vigo (eds.), "*The Vehicle Routing Problem*", SIAM Monographs on Discrete Mathematics and Applications, Philadelphia (2002).