Solving Vehicle Routing Problems Using Constraint Programming and Lagrangean Relaxation in a Metaheuristics Framework

D. Guimarans*, R. Herrero, J.J. Ramos, S. Padrón
Dpt. Telecommunication and Systems Engineering,
Universitat Autònoma de Barcelona, Spain

ABSTRACT

This paper presents a methodology based on the Variable Neighbourhood Search metaheuristic, applied to the Capacitated Vehicle Routing Problem. The presented approach uses Constraint Programming and Lagrangean Relaxation methods in order to improve algorithm's efficiency. The complete problem is decomposed into two separated subproblems, to which the mentioned techniques are applied to obtain a complete solution. With this decomposition, the methodology is able to provide a quick initial feasible solution which is rapidly improved by metaheuristics' iterative process. Constraint Programming and Lagrangean Relaxation are also embedded within this structure to ensure constraints satisfaction and to reduce the calculation burden. By means of the proposed methodology, promising results have been obtained. Remarkable results presented in this paper include a new best-known solution for a rarely solved 200-customers test instance, as well as a better alternative solution for another benchmark problem.

Keywords: Vehicle Routing, Constraint Programming, Lagrangean Relaxation, Metaheuristics, Variable Neighbourhood Search, Hybrid Algorithms.

INTRODUCTION

Routing vehicles to collect or delivery goods is a problem which many companies face each day, laying at the heart of many distribution systems. In practice, objectives and constraints are highly variable and, most of times, complex. In fact, real problems often require a specific modelling and solving methodology. On the other hand, most research is focused on well-known sets of academic problems including certain characteristics. However, since flexible and efficient algorithms are likely to be adapted to various practical contexts, these prototype problems become a nice reference where to test developed methodologies.

This class of logistics problems, usually known as the Vehicle Routing Problem (VRP), is among the most popular research areas in combinatorial optimization. Since it was first defined by Dantzig and Ramser (1959), several variants of the basic problem have been proposed and studied. The most basic VRP is the Capacitated Vehicle Routing Problem (CVRP) that assumes a fleet of vehicles with homogeneous capacity housed in a single depot. It is so a generalization of the Travelling Salesman Problem (TSP) and is therefore NP-hard (Savelsbergh, 1985). For such problems, finding an optimal solution requires a high computational effort.

Several formulations and exact algorithms have been proposed to solve the CVRP. However, for large instances the time required to solve them becomes absolutely prohibitive due to its NP-hardness. Thus, exact algorithms may only deal with small instances, up to 100 customers (Cordeau et al., 2007), solving them to optimality. Numerous heuristics and metaheuristics have also been studied for different VRP variants. In most cases, these methods may solve larger instances but loosing optimality guarantees. This field has deserved special attention from the research community and has stimulated the emergence and the growth of several metaheuristics of general applicability. A recent overview of available methods for different VRP variants can be found in Cordeau et al. (2007).

Compared with classical heuristics, such as route construction and improvement methods or two-phase approaches, metaheuristics are less likely to end trapped in a local optimum. Results justify community's interest in this field. As an example, the large number of Tabu Search based algorithms produced over the last years (Cordeau & Laporte, 2004) is remarkable. Among metaheuristics, Variable Neighbourhood Search (VNS), introduced for the first time in Mladenovic and Hansen (1997), is a quite recent method with far less application examples in VRP research. However, interesting results have been obtained even applying the simplest VNS algorithm, the Variable Neighbourhood Descent (VND) (Bräysy, 2003; Hasle & Kloster, 2007; Rousseau et al., 2002). For this reason, VNS has been selected as the general framework where to embed Constraint Programming (CP) and Lagrangean Relaxation (LR) approaches to the CVRP. By using these two well-known paradigms within the VNS local search process, calculation time may be reduced with respect to classical VNS schemes. Such a hybrid methodology has been adopted as a first approach, suitable to be modified and improved in order to tackle more complex VRP variants, i.e. VRP with Time Windows (VRPTW) and the Pick-Up and Delivery VRP with Time Windows (PDTW). With this objective, the CVRP has been chosen to test algorithm's effectiveness and major efforts have been addressed to obtain good quality solutions rather than low computation times. Thus, the presented approach becomes a necessary first step to analyse other VRP categories, for which main guidelines introduced in this paper still hold.

This paper is aimed to present a general VNS structure whose local search process is based on CP and LR. The CVRP is divided into two separate subproblems which are modelled and solved using mentioned paradigms. The present paper explains the chosen decomposition and how separate problems are tackled in terms of CP and LR approaches. Promising results, both for finding a quick initial solution and for solving the whole problem, are also shown. Remarkable results presented in this paper include a new best-known solution for a rarely solved 200-customers benchmark problem, as well as an alternative solution, with a lower cost and using one vehicle less, for a well known one.

The remainder of this article is structured as follows. Next section provides a general overview of CVRP formulation, emphasizing the decomposition used in the proposed method. An introduction to Lagrangean Relaxation is presented afterwards. The following section is devoted to the proposed method, based on the VNS metaheuristic; the general algorithm, moves used within its structure and the adapted LR-method are introduced in this section. Next, computational results are presented and discussed. Finally, some conclusions, remarks and future research topics are outlined in the last section.

PROBLEM FORMULATION

The symmetric CVRP can be considered as a complete undirected graph $G=(I,E)$, connecting the vertex set $I=\{1, 2, ..., n\}$ through a set of undirected edges $E=\{(i,j) \mid i,j \in I\}$. The edge $e_{ij} \in$ E has associated a travel cost c_{ij}, supposed to be the lowest cost route connecting node i to node j. Each vertex $i \in I\backslash\{1\}$ has a nonnegative demand q_i, while vertex 1 corresponds to a depot without associated demand. A fixed fleet of m identical vehicles, each of capacity Q, is available at the depot to accomplish the required tasks.

Solving the CVRP consists of determining a set of m routes whose total travel cost is minimised and such that: (a) each customer is visited exactly once by a single vehicle, (b) each route starts and ends at the depot, and (c) the total demand of the customers assigned to a route does not exceed the vehicle capacity Q. Therefore, a solution to the CVRP is a set of m cycles sharing a common vertex at the depot. In some cases, the fleet size is not fixed and minimising the total number of used vehicles becomes an additional objective.

In the proposed model, the CVRP has been divided into two subproblems, concerning customers' allocation and routing optimization separately. The first is aimed to assign customers to vehicles fulfilling capacity limitations. The latter is used to solve each independent route to optimality, giving the best solution for a particular allocation. Thus, routing optimization process can be viewed as solving a set of m independent symmetric TSP. CP is used to find a feasible solution in terms of capacity, while routing problems are solved by using LR.

Capacity problem

Constraint Programming is a powerful paradigm for representing and solving a wide range of combinatorial problems. Problems are expressed in terms of three entities: variables, their corresponding domains and constraints relating them. The problems can then be solved using complete techniques such as depth-first search for satisfaction and branch and bound for optimization, or even tailored search methods for specific problems. Rossi et al. (2006) presents a complete overview of CP modelling techniques, algorithms, tools, and applications.

The proposed capacity subproblem uses the following variables:

- $R = R_1, ..., R_n$ with an integer domain $[1..m]$
- $Q_v = Q_1, ..., Q_m$ with a real domain $[0..Q]$

R is a list of n variables, corresponding to the n customers. Each R_i value indicates which vehicle is serving the i^{th} customer, and so it can take values from 1 to m. R_1 value is not relevant, since it corresponds to the depot and has no associated demand. However, it is included for simplicity reasons on defining variables lists.

Q_v is a list of m variables used to trace the cumulative capacity at each of the m routes. Capacity constraints are enforced through domains definition, forcing each Q_v to take values up to a maximum corresponding to vehicles' capacity Q.

A set of dimension $m \times n$ of binary variables B has been introduced to relate R and Q_v values. For each vehicle v ($v=1, ..., m$), a list of n binary variables B_{vi} ($i=1, ..., n$) is defined, taking value 1 whenever customer i is assigned to vehicle v and 0 otherwise. Since each customer i is visited by a single vehicle, for all values of v the binary variable B_{vi} can take value 1 only once. This constraint is expressed in terms of the global constraint *occurrences*, included as a built-in predicate in the software ECLiPSe (Apt & Wallace, 2007), a specific platform aimed to

developing and solving CP models programmed in Prolog, and suitable to be linked with external applications.

$$occurrences(1, B_{vi}, 1) \quad \forall v \in [1, m] \tag{1}$$

Expression (1) states that value *1* can occur only once in the list of variables B_{vi}, i.e. for a fixed value *i*, only one of the *m* elements of the list B_{vi} can take value 1. Predicate *occurrences* may be seen as an implementation of the general global constraint *cardinality* (Beldiceanu et al., 2005).

Using global constraints increases the search efficiency. Whenever a variable is instantiated during the search process, propagation mechanisms reduce uninstantiated variables' domains to some degree (Bessiere, 2006). Global constraints ensure a faster reduction of domains through specifically programmed propagation methods. Moreover, they allow a clean and fast definition of constraints patterns for sets of variables of any size.

The binary set *B* and allocation variables *R* are related through the following statement:

$$R_i = r_i \rightarrow B_{r_i i} = 1 \quad \forall i \in I \tag{2}$$

Expression (2) states that the i^{th} element of the r_i list of *B* will have value 1 whenever the i^{th} component of *R* takes value r_i. Global constraint (1) ensures propagation so all values of $B_{vi} \mid v \in \{1,...,m\} \setminus r_i$ are set to 0 automatically. Therefore, cumulative capacities can be traced simply by using the following equation:

$$Q_v = \sum_{i \in I} B_{vi} q_i \quad \forall v \in V \tag{3}$$

The proposed formulation is used to find a partial initial solution fulfilling capacity constraints. By solving resultant routing problems, which are always feasible because they do not contain any additional constraints, a complete initial solution may be easily obtained in most cases. Thus, capacity problem's goal is to find a feasible solution with the minimum number of required vehicles. With this objective, a depth-first search method is applied to find a feasible solution that uses all available vehicles. A vehicle is removed from the list and the process is repeated recursively. The algorithm stops when unfeasibility is reached, returning the last feasible solution found in the previous iteration.

Routing problem

The routing problem, tackled for each vehicle separately, can be viewed as a TSP instance. The TSP is probably the best known combinatorial problem: “A salesman is required to visit once and only once each of *n* different customers starting from a depot, and returning to the depot. What path minimises the total distance travelled by the salesman?” (Bellman, 1962).

For each vehicle *v*, the related TSP can be considered as a complete undirected graph $G=(I_v,E_v)$, connecting assigned customers $I_v=\{i \in I \mid R_i=v\}$ through a set of undirected edges $E_v=\{(i,j) \in E \mid i,j \in I_v\}$. The solution is a path connected by edges belonging to E_v that starts and ends at the depot ($i=1$) and visits all assigned customers.

Then a feasible solution of the TSP should, by definition, also satisfy constraints (a) and (b) of the CVRP, minimising the total travel cost of the route.

The proposed mathematical formulation requires defining the binary variable x_e to denote that the edge e_{ij} *in* E_v is used in the path:

$$x_e = \begin{cases} 1 & \text{if customer } j \text{ is visited immediately after } i; \\ 0 & \text{otherwise.} \end{cases}$$

The proposed mathematical formulation for the TSP problem is as follows:

$$\min \sum_{e \in E_v} c_e x_e \tag{4}$$

subject to

$$\sum_{e \in \delta(i)} x_e = 2, \quad \forall i \in I_v \tag{5}$$

$$\sum_{e \in E_V(V)} x_e \le |V| - 1, \quad \forall V \subset I_v, |V| \le \frac{1}{2} n_v \tag{6}$$

where

- $\delta(i) = \{e \in E_v \mid \exists j \in I_v, e = (i,j) \text{ or } (j,i)\}$ represents the set of arcs whose starting or ending node is i.
- $E_v(V) = \{e = (i,j) \in E_v \mid i,j \in V\}$ represents the set of arcs whose nodes are in the subset of vertices V.
- $n_v = |I_v|$

The objective function (4) aims to minimise the total cost of the route. Constraint (5) states that every node $i \in I_v$ must be visited once. Since c_e is the associated cost to the undirected edge e_{ij} (e_{ji}), every customer must have two incident edges. Subtour elimination constraint (6) states that the tour must be a Hamiltonian cycle, e.g. for every pair of vertices there is a path connecting them, so it can not have any subcycle. This constraint avoids any subcycle of the subset V, since the number of edges must be lower than the size of the subset. It only considers the subset V which $|V| \le \frac{1}{2} n_v$. For any solution containing more than one subcycle, at least one of them will fulfil $|V| \le \frac{1}{2} n_v$. If these are banned, all subcycles are avoided.

LAGRANGEAN RELAXATION

Lagrangean Relaxation is a well-known method to solve large-scale combinatorial optimization problems. It works by moving hard-to-satisfy constraints into the objective function associating a penalty in case they are not satisfied. An excellent introduction to the whole topic of LR can be found in Fisher (1981). For a recent review, see Guignard (2003).

LR exploits the structure of the problem, so it reduces considerably problem's complexity. Thus, the Lagrangean Problem needs less computational effort to be solved. However, it is often a major issue to find the optimal Lagrangean multipliers. The commonly used approach is the Subgradient Optimization method. It guarantees convergence, but it is too slow to become a method of real practical interest.

The proposed LR-based method improves the convergence on the optimal solution of the Subgradient Optimization by using a heuristic to obtain a feasible solution from a LR solution. If the optimal solution is not reached at a reasonable number of iterations, the proposed method is able to provide a feasible solution with a tight gap between the primal and the optimal cost.

Given the assigned customers to each vehicle, the proposed LR-based method is used to solve associated routing problems. In this proposed approach, LR relaxes the constraint set requiring that all customers must be served (5), since all subcycles can be avoided constructing the solution *x* as a 1-tree. Actually, a feasible solution of the TSP is a 1-tree having two incident edges at each node. A 1-tree can be defined as a tree on the graph induced by nodes $\{2,...,n_v\}$ plus two incident edges at node 1 (Held & Karp, 1971).

The Lagrangean Dual problem obtained from the TSP formulation, moving into the objective function equalities (5), is:

$$\max_{u \in \Re^{n_v}} L(u) \tag{7}$$

where the Lagrangean function is:

$$L(u) = \min_{x\ 1-tree} \sum_{e \in E_v} c_e x_e + \sum_{i \in I_v} u_i \left(2 - \sum_{e \in \delta(i)} x_e \right) \tag{8}$$

This LR relaxes constraints (5) weighting them with a multiplier vector *u* of appropriate dimension and unrestricted sign, defining the subgradient $\gamma_i = 2 - \sum_{e \in \delta(i)} x_e$.

Finding a minimum spanning tree induced by nodes $\{2,...,n_v\}$ is relatively easy, so the presented relaxation becomes potentially interesting. Prim's Algorithm is commonly used for finding a minimum spanning tree (Laporte, 1992). Its time complexity is $O(N^2)$, where $N=n_v-1$ is the number of vertices.

The Subgradient Optimization is an iterative method used to solve the Lagrangean problem finding a maximum value of the lower bound (Held et al., 1974). The main difficulty of this algorithm lays on choosing a correct step-size λ_k (Wolsey, 1998). This is a critical choice, since the convergence can be highly influenced by this parameter (Reinelt, 1994).

Being *LB* a dual lower bound and L^* the optimal value, so $LB \le L^*$, the step-size is defined according to $\lambda_k = \delta_k \dfrac{LB - L(u^k)}{\left\| \gamma^k \right\|^2}$ with $0 < \delta_k \le 2$. Then, $L(u^k) \to LB$, or the algorithm finds u^k with $LB \le L(u^k) \le L^*$ for some finite *k*. In practice, *LB* is typically unknown and it is more likely to know a good primal upper bound $UB \ge L^*$. Such an upper bound *UB* is then used initially in place of *LB*. However, if $UB >> L^*$, the term $UB\text{-}L(u^k)$ in the numerator will not tend to zero, and

so sequences $\{u^k\}$ and $\{L(u^k)\}$ will not converge. In order to find a feasible solution of the TSP, which may give an accurate *UB*, a Nearest Neighbour Heuristic is applied. This method is commonly used with this purpose, since it is computationally efficient and easy to implement.

SOLUTION METHODS

The described problem has been tackled using a hybrid approach. The proposed methodology combines CP and LR within a metaheuristics framework in order to improve algorithm's performance. As mentioned, even the most basic Variable Neighbourhood Search algorithm, known as Variable Neighbourhood Descent (VND), has provided promising results when solving different VRP variants. In the proposed approach, a general Variable Neighbourhood Search (VNS) framework has been chosen to embed selected paradigms. A complete and revised description of different VNS algorithms can be found in (Hansen & Mladenovic, 2003).

In any case, other well known metaheuristics could have been used to embed CP and LR, such as Tabu Search (TS) or Genetic Algorithms (GA). Both metaheuristics have been widely used for tackling different VRP variants obtaining good results (Bräysy & Gendreau, 2005). However, VNS permits overcoming some of their limitations. On the one hand, TS is based on a local search where the process may lead to worse solutions in order to escape from local minima. It may be comparable to the VND algorithm, but the latter has the advantage of alternating different moves to explore the search space. Swapping these neighbourhoods structures allows escaping from local minima in a more natural manner and avoids defining and tuning tabu lists and aspiration criteria. Moreover, by using a general VNS scheme, a diversification process is naturally introduced and integrated within the algorithm, so search is restarted at each iteration from a point obtained from the best solution found so far. This diversification process may be tuned so the algorithm behaves conservatively or following a multistart strategy. On the other hand, GA requires defining some parameters that become critical for algorithm's performance, such as the population size or crossover and mutation ratios. According to problem dimensions, managing correctly the memory used to store population data may become a major issue. In some cases, reducing the population size may lead to misleading results (Reeves, 2003). Thus, finding a trade-off between efficiency and effectiveness often needs a fine-tuning process that is not required when using a VNS algorithm. For these reasons, the general VNS has been selected as the main structure that leads the search process in the proposed methodology.

Within the general VNS framework, CP and LR are used in different processes. During algorithm's initialization, CP is used to find an initial feasible solution by means of capacity constraints. CP is also used to check solutions feasibility within diversification and local search processes.

In turn, a tailored LR method is applied to calculate routes every time a partial solution is generated either during initialization, diversification or local search processes. Using LR allows reducing the computation time when compared to other routing post-optimization methods, such as a VND with single-route classical moves. So, the proposed LR approach provides optimal routes in very low times and, at the same time, permits reducing algorithm's definition and complexity.

Proposed Lagrangean Relaxation method

The proposed LR-based method is used within the local search process to solve routing subproblems to optimality. It can be considered a specification of the Lagrangean Metaheuristic

presented on Boschetti & Maniezzo (2009). It uses the Subgradient Optimization algorithm combined with a heuristic. Aiming to improve algorithm's convergence to the optimal solution, a heuristic is introduced in order to obtain a feasible solution from the dual variable. This method tries to improve the *UB* with the values of these feasible solutions, so a better convergence is obtained. Eventually, this feasible solution may be provided as the best solution if the method is stopped. The stopping criterion is based on the maximum number of iterations ($k < max_{iterations}$) and also on a floating-point exception ($\lambda_k < 10^{-15}$). The proposed LR-based method is shown in Algorithm 1.

Algorithm 1: The Proposed LR-based Method

0 Initialization

1 Initialize parameters $u^0 \equiv 0;\ \delta_0 = 2;\ \rho = 0.95;\ \alpha_L = 1/3$

2 Obtain an UB applying Nearest Neighbour Heuristic

3 Initialize $\overline{L} = L(u^0) + \alpha_L (UB - L(u^0))$

4 Iteration k

5 Solve the Lagrangean function $L(u^k)$

6 Check the subgradient $\gamma_i^k = 2 - \sum_{e \in \delta(i)} x_e$

7 if ($\left\|\gamma^k\right\|^2 = 0$) then Optimal solution is found $\Rightarrow$ EXIT

8 if ($\left\|\gamma^k\right\|^2 < \zeta$) then apply a heuristic to improve the UB

9 Check the parameter $\overline{L}$

10 Calculate the step-size $\lambda_k = \delta_k \dfrac{\overline{L} - L(u^k)}{\left\|\gamma^k\right\|^2}$

11 Update the multiplier $u^{k+1} = u^k + \lambda_k \gamma^k$

12 $k \leftarrow k + 1$

The proposed heuristic to improve the *UB* is applied when the solution is nearly a route. That is, if it satisfies $\left\|\gamma^k\right\|^2 < \zeta$ (step 8). As any solution is a 1-tree, this criterion means that the solution has few vertices without two incident edges. This heuristic replaces an edge e_{ij} where j has some extra edges for an edge e_{il} where l has one single edge. Before applying the exchange, the heuristic checks if the new solution is a 1-tree. Otherwise, the heuristic can divide it into more trees having some subtours. The chosen vertices i,j,l minimise the cost of the exchange:

$$\{i, j, l\} = \arg\min \left\{ c_{il} - c_{ij} : \gamma_j < 0,\ \gamma_l > 0,\ \gamma_i \le 0,\ x_{ij} = 1,\ x_{il} = 0 \right\} \tag{9}$$

The parameter ζ depends on the number of variables. A good estimation of ζ value would avoid increasing the computation time. First, its value may be large, for instance $n\sqrt{2}$, but it should be updated whenever a feasible solution is found according to $\zeta = \left\|\gamma^k\right\|^2$. If this

parameter is not correctly updated, the heuristic becomes time consuming. Eventually, the heuristic could find the optimal solution without detecting it, so the method would continue iterating until LB=UB.

As mentioned, algorithm's convergence is critically influenced by the step-size λ_k. This value relies on either the *LB* or the *UB*, which are normally unknown or bad estimated. Therefore, convergence may not be assured for all cases. In order to overcome this limitations, the use of a parameter $\overline{L}$, such that $LB \le \overline{L} \le UB$, is proposed. By definition, this parameter corresponds to a better estimation of the optimum L^* than those obtained for *LB* and *UB*. The calculation of the step-size turns into:

$$\lambda_k = \delta_k \frac{\overline{L} - L(u^k)}{\left\| \gamma^k \right\|^2} \tag{10}$$

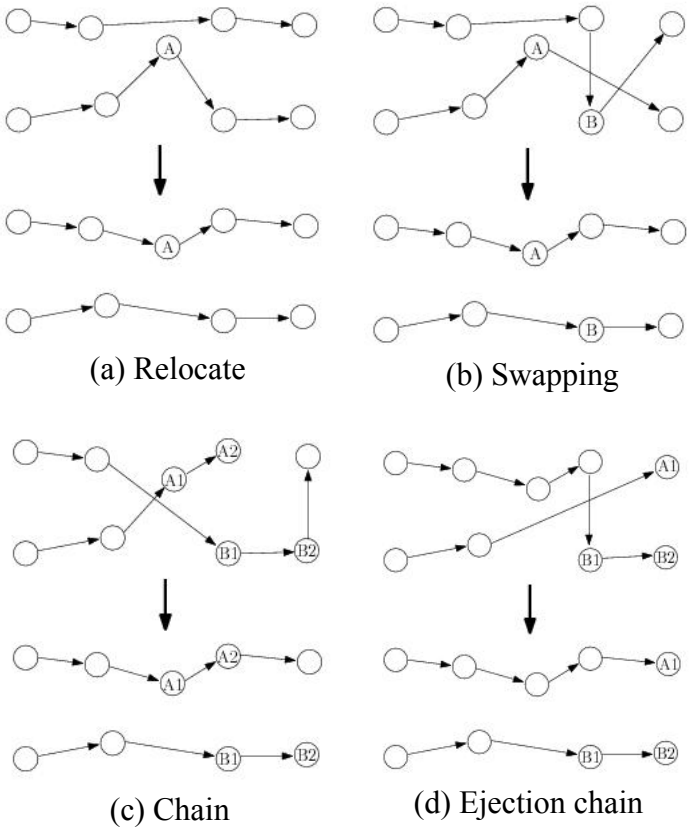
Convergence is guaranteed if the term $\overline{L} - L(u^k)$ tends to zero. In turn, convergence efficiency can be improved as long as the new $\overline{L}$ parameter gets closer to the (unknown) optimal solution. The main idea is very simple: as the algorithm converges to the solution, new better lower bounds are known and new better upper bounds estimations can be obtained by using the heuristic designed to get feasible solutions. Therefore, the parameter $\overline{L}$ is updated according to the following conditions:

- It is initialized $\overline{L} = L(u^0) + \alpha_L (UB - L(u^0))$ with $0 < \alpha_L < 1$.
- If $L(u^k) > \overline{L}$, it is updated $\overline{L} = L(u^k) + \alpha_L (UB - L(u^k))$.
- If $\overline{L} > UB$, finally $\overline{L} = UB$.

Finally, the parameter δ_k is initialized to the value 2 and is updated as Zamani & Lau (2010) suggest. If the lower bound is not improved, δ_k is decreased, using the formula $\delta^{k+1} = \delta^k \rho$ with $0 < \rho < 1$. On the other hand, if the lower bound is improved, then its value is increased according to the formula $\delta^{k+1} = \delta^k \frac{3-\rho}{2}$, providing that $0 \le \delta^k \le 2$ to ensure convergence.

Inter-route moves

VNS metaheuristic is based on exploring alternatively different neighbourhoods around a known feasible solution. In order to establish these neighbourhoods, different moves are to be defined. In the presented approach, four different inter-routes classic moves (Savelsbergh, 1988) have been defined so they can be used within diversification and local search processes (Figure 1):



(a) Relocate (b) Swapping

(c) Chain (d) Ejection chain

Figure 1. Inter-route movements.

- *Relocate*: moves a customer from one route to a different one.
- *Swapping*: exchanges two customers belonging to two different routes.
- *Chain*: is a specialization of 3-opt that swaps sections of two contiguous customers from two different routes.
- *Ejection chain*: swaps the end portions of two different routes. In the implemented approach, the relative percentage of customers modified has been arbitrarily set to 40 %.

As mentioned, using LR for solving the routing subproblems allows avoiding the definition of intra-route moves. Since results provided by the LR method are optimal, no routing optimization process is needed. Usually, a post-optimization method based on intra-route moves is applied to improve each single route quality (Rousseau et al., 2002).

Variable Neighbourhood Search framework

A general VNS framework, as explained in Hansen & Mladenovic (2003), has been implemented embedding the described methods. A simplified scheme of the method is presented in Algorithm 2. At each iteration, a local minimum is reached departing from an initial solution. A diversification process (shaking) ensures that different regions from the search space are explored by changing the initial solution at each iteration.

Algorithm 2: Variable Neighbourhood Search

0 Initialization. Select the set of neighbourhood structures N_k, for $k=1,...,k_{max}$, that will be used in the shaking phase, and the set of neighbourhood structures N_l for $l=1,...,l_{max}$ that will be used in the local search; find an initial solution x; choose a stopping condition;

1 Repeat the following sequence until the stopping condition is met:

2 Set $k \leftarrow 1$;

3 Repeat the following steps until $k = k_{\max}$:

4 (a) Shaking. Generate a point x' at random from the k^{th} neighbourhood $N_k(x)$ of x;

5 (b) Local search by VND.

6 (b1) Set $l \leftarrow 1$;

7 (b2) Repeat the following steps until $l = l_{\max}$;

8 - Exploration of neighbourhood. Find the best neighbour x'' of x' in $N_l(x')$;

9 - Move or not. If $f(x'')<f(x')$ set $x' \leftarrow x''$ and $l \leftarrow 1$; otherwise set $l \leftarrow l+1$;

10 (c) Move or not. If this local optimum is better than the incumbent, move there ($x \leftarrow x''$), and continue the search with N_1 ($k \leftarrow 1$); otherwise, set $k \leftarrow k+1$;

In step 0, neighbourhood structures to be used within shaking (N_k) and local search (N_l) processes are selected. In the proposed implementation, all four described moves have been selected to be used in both neighbourhoods. As a first step in the algorithm, an initial feasible solution is found using CP and LR. CP is used to assign all customers to available vehicles fulfilling capacity constraints, while resulting routes are solved to optimality by means of LR. Finally, the stopping criterion is chosen to be based on the maximum number of iterations.

In step 4, a new point is generated at random from the k^{th} neighbourhood $N_k(x)$ of x in order to diversify the search. Its feasibility is immediately checked using CP. If the generated point is unfeasible, the process is repeated until a new feasible point is found. However, if the valley surrounding the solution x is large, a thorough diversification should be done aiming to avoid getting trapped in a local optimum. For this reason, the implemented shaking process is repeated several times. Solutions' values are ignored until the last iteration, when routes are recalculated using LR to provide a complete solution.

Steps 5 to 9 contain the VND algorithm used to perform the local search. It should be remarked that it takes advantage of all neighbourhood structures for $l=1,...,l_{max}$, instead of applying a simple local search method. In any case, it may also get trapped in a local optimum. For this reason, VND is used within a diversification/local search iterative process (steps 3 to 10).

Within the VND algorithm, an exhaustive exploration of the l^{th} neighbourhood $N_l(x')$ of x' is performed in step 8. Departing from the solution x', the l^{th} move is applied and new solution's feasibility is checked using CP. Whenever it is proved feasible, LR is used to recalculate only modified routes. This approach permits to consider only two routes per solution, reducing the

computation time. Finally, the best neighbour *x''* is chosen in terms of its solution value $f(x'') = \sum_{v=1}^{m} UB_v$.

A slightly different approach has also been implemented for the exhaustive exploration performed at step 8. In this case, CP consistency techniques (Bessiere, 2006) are applied to get feasible domains for the variables to be modified. Thus, only values corresponding to feasible solutions are explored and capacity constraints do not need to be checked afterwards. This approach provides the same results, but the calculation time is dramatically increased due to consistency algorithms' high complexity. However, tailored propagation and consistency techniques could lead to an important time reduction and so it is a promising line for future research work.

COMPUTATIONAL RESULTS

The methodology described in the present paper has been implemented in Java and linked to the open-source CP software system ECLiPSe 6.0 (Apt & Wallace, 2007). All tests have been performed on a non-dedicated server with an Intel Xeon Quad-Core i5 processor at 2.66GHz and 16GB RAM. In general, five to seven processes were launched in parallel to solve different problems, while external applications were active at the same time. For this reason, CPU times obtained with the proposed methodology are to be considered as approximated.

A total of 97 classical CVRP benchmark instances have been used to test the efficiency of the proposed approach. They have been obtained from (Branchandcut.org, 2003), a reference site with a large number of benchmark sets for different combinatorial problems, yet not updated results. Best known solutions for problems not solved to optimality have been updated with recent references in order to provide a thorough comparative with results obtained by using the presented methodology. Instances were selected according to the distance type used in their definition. Only those instances whose distance is defined as Euclidean or Geographic have been selected, in order to ensure triangular inequality's fulfilment. Therefore, all problems from benchmark sets A, B, F, G, M, and P have been included. In addition, those instances from the set E accomplishing the mentioned criterion are also considered, as well as 3 TSPLib (Reinelt, 2008) converted problems (att-n48-k4 and both ulysses instances).

Distances have been rounded to integers, according to the specification included in the TSPLib. This approach allows comparing results obtained with those published in a wide range of references working over the same benchmark sets. However, the proposed approach is not restricted to work with integer distances, so it can be used without adding any modification to solve same instances considering real costs. Although a good number of papers adopt this realistic approach, results presented in this paper are aimed to be compared with best known integer solutions. In order to compare obtained solutions with best known realistic ones, the algorithm should be run using real costs matrices, since rounding distances prior to solving make both instances different, and so their corresponding solutions.

In all tests, *swapping* has been set as the initial move both for shaking (*k=1*) and local search (*l=1*) processes. *Relocate*, *chain*, and *ejection chain* are used next whenever the previous solution is not improved. A summary of obtained results with this neighbourhood set is presented in Table 1. Another test has been done exchanging *swapping* and *relocate* priorities, getting a similar performance. For this reason, only results from the first configuration are shown. However, it is remarkable that, in general, the *swapping*/*relocate* configuration has a better

performance when applied to class M problems, while *relocate*/*swapping* behaves much better on class B instances. Probably, performance's differences rely on customers' distribution and a further revision is to be done in a near future.

Table 1 presents the number of problems successfully solved to optimality by using the proposed methodology, as well as the number of problems whose optimal value was not reached and those which could not be solved. Table 1 also shows the average (% Dev.), maximum (% Max) and minimum (% Min) deviation from the best known value for those problems that could not be solved to optimality. A low deviation is observed for most problem sets, comparable to results obtained by means of other metaheuristics.

Class	Problems	Opt.	No opt.	Not solved	% Dev.	% Max	% Min
A	27	14 (51,85%)	13 (48,15%)	0 (0%)	0,65	2,14	0,14
B	23	12 (52,17%)	11 (47,83%)	0 (0%)	1,79	4,28	0,13
E	11	5 (45,45%)	6 (54,55%)	0 (0%)	0,68	1,59	0,41
F	3	1 (33,33%)	2 (66,67%)	0 (0%)	4,22	4,22	4,22
G	1	0 (0%)	1 (100%)	0 (0%)	0,44	0,44	0,44
M	5	1 (20%)	4 (80%)	0 (0%)	2,44	4,35	0,69
P	24	14 (58,33%)	8 (33,33%)	2 (8,33%)	0,88	3,3	0,15
TSPLib	3	1 (33,33%)	2 (66,67%)	0 (0%)	2,28	3,23	1,33
	97	48 (49,48%)	47 (48,45%)	2 (2,06%)	1,67	2,94	0,94

Table 1. Summary of results obtained with a swapping/relocate configuration.

As mentioned, the initial solution is obtained by solving separately capacity and routing problems. This approach is able to provide a low-quality quick solution, since both subproblems are easily solved but variables are unlinked. However, this solution is highly improved at the first iteration. As an example, this approach may provide an initial solution for larger problems, such as M-n200-k16, in less than 6 seconds. After the first iteration, its value is usually close to the final result.

Furthermore, the use of LR ensures the partial optimality of all solutions from the routing perspective. The reason is that the proposed LR approach can optimally solve all TSP instances. As can be seen in Figure 2, LB and UB converge rapidly. For all problems, their gap is always located between 0 and 10^{-10}, guaranteeing so the solution optimality. Moreover, LR solves all routes in negligible times, due to the number of associated customers is always low. Thus, LR has demonstrated to be an efficient alternative for intra-route optimization processes.

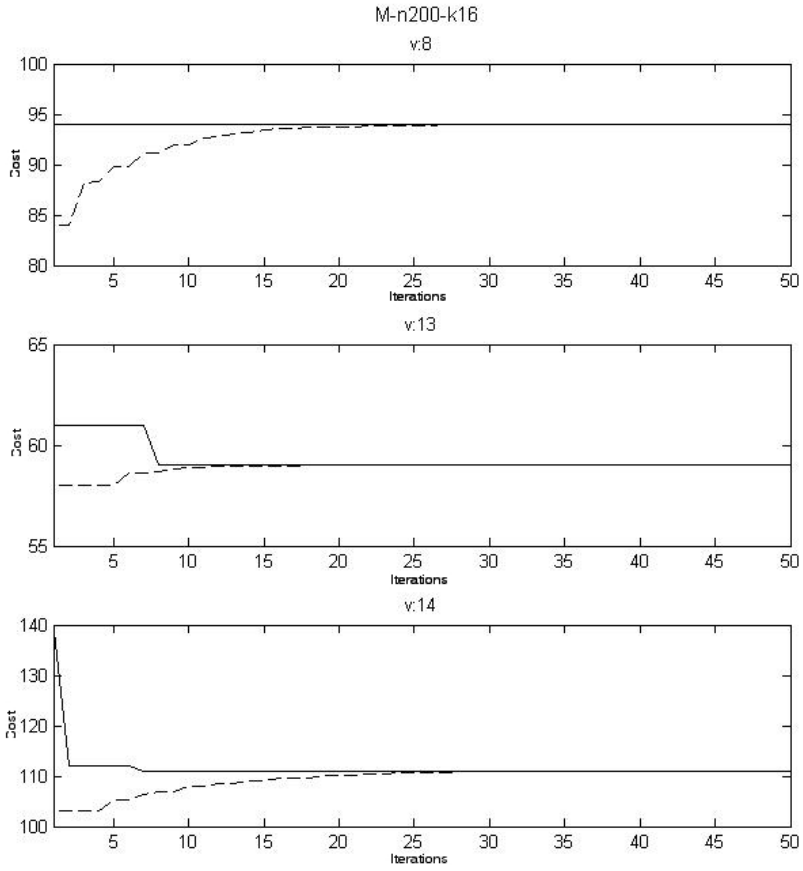


Figure 2. Convergence of LB (dashed line) and UB (solid line) in three routes from problem M-n200-k16. Although LR maximum number of iterations is set to 300, it usually converges in less than 50 iterations for most problems.

The detailed behaviour of the presented method can be observed in Figure 3. The solid line corresponds to solution's evolution, while the dotted line shows algorithm's behaviour at each iteration. Every shaking process draws a peak, while the local search process leads the algorithm to a solution with a lower cost. It can be observed that, after some iterations, the local search process converges more often to the best solution so far. As the algorithm evolves, so it does solution's structure quality, getting closer to the optimum. Therefore, a more thorough shaking process would be needed to get the solution far enough so different regions from the solution space were explored. However, it could cause the algorithm to diverge. As an example, when *chain* and *ejection chain* are applied in the shaking process, the local search process either converges slowly or may not find the best solution so far. Since both moves modify larger sections than *relocate* or *swapping*, the exploration may be leaded to regions far from the optimal.

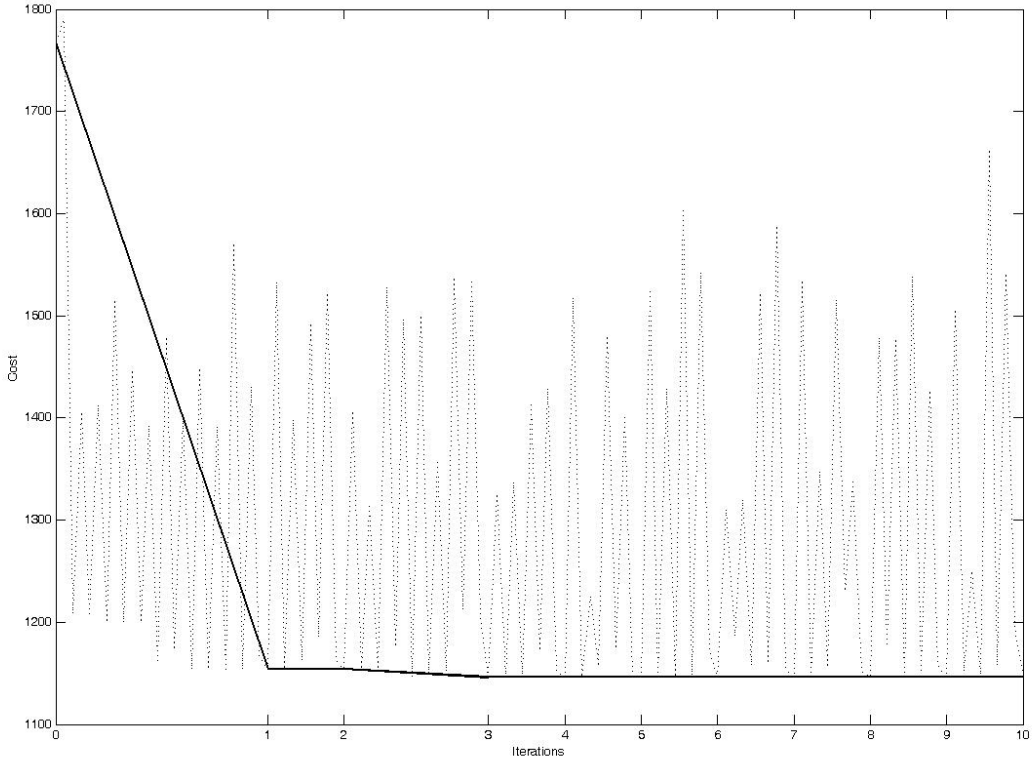


Figure 3. An example of the local search process on problem A-n45-k7 for first 10 iterations. The solid line corresponds to final solution evolution, while the dotted line shows the shaking process and the local search behaviour.

The proposed methodology performs similarly both for small and large instances. Thus, its applicability is not restricted. It is remarkable that the algorithm eventually reaches the optimal solution for smaller problems (50 customers or less), but it stops near the optimum for larger instances. Tables 2 and 3 present representative results obtained for classes A and P. These results are similar to those obtained for remaining classes. Tables 2 and 3 compare obtained values to best known solutions (columns *BKS* and *Opt.#veh.*). They show the initial and the final solution, computational times spent on calculating them, and corresponding gap of the final solution. The computational time needed to reach a solution with a gap lower than 2 %, selected as a good quality solution threshold, has also been included in all tables. Last column presents the iteration where the best-known value is reached or improved, where value -1 indicates that the BKS has not been obtained.

Table 4 shows that the presented approach is able to provide state-of-art results for large benchmark instances (100 customers or more). As observed, final values are normally close to best known solutions. It is also remarkable the result obtained for the largest selected test instance G-n262-k25, which stays slightly (0.44 %) over the best known value presented in (Hasle & Kloster, 2007).

Problem	#Cust.	Veh.Cap.	BKS	Init.Sol.	CPU_{IS} (s)	Fin.Sol.	CPU_{FS} (s)	Gap BKS-FS (%)	$CPU_{gap\leq 2\%}$ (s)	#Veh. / Opt.#Veh.	Opt.It.
A-n32-k5	31	100	784	1243	0.256	784	96.62	0.00 %	96.62	5 / 5	2
A-n33-k5	32	100	661	1185	0.018	661	42.67	0.00 %	28.05	5 / 5	2
A-n33-k6	32	100	742	1289	0.013	742	34.86	0.00 %	34.86	6 / 6	32
A-n34-k5	33	100	778	1259	0.014	778	27.88	0.00 %	27.88	5 / 5	1
A-n36-k5	35	100	799	1207	0.024	799	524.33	0.00 %	148.16	5 / 5	22
A-n37-k5	36	100	669	960	0.012	669	65.76	0.00 %	19.82	5 / 5	2
A-n37-k6	36	100	949	1393	0.020	949	59.67	0.00 %	27.10	6 / 6	14
A-n38-k5	37	100	730	1240	0.017	731	69.02	0.14 %	23.07	5 / 5	-1
A-n39-k5	38	100	822	1291	0.010	822	277.41	0.00 %	39.54	5 / 5	8
A-n39-k6	38	100	831	1523	0.014	833	554.50	0.24 %	84.94	6 / 6	-1
A-n44-k6	43	100	937	1547	0.020	942	149.39	0.53 %	60.68	6 / 6	-1
A-n45-k6	44	100	944	1826	0.022	950	141.67	0.64 %	64.16	6 / 6	-1
A-n45-k7	44	100	1146	1768	0.022	1146	207.30	0.00 %	77.42	7 / 7	3
A-n46-k7	45	100	914	1711	0.024	914	265.87	0.00 %	94.97	7 / 7	2
A-n48-k7	47	100	1073	1840	0.025	1084	295.10	1.03 %	178.55	7 / 7	-1
A-n53-k7	52	100	1010	1841	0.030	1020	1291.31	0.99 %	122.65	7 / 7	-1
A-n54-k7	53	100	1167	1883	0.051	1167	321.39	0.00 %	202.48	7 / 7	8
A-n55-k9	54	100	1073	2074	0.034	1073	1387.84	0.00 %	218.51	9 / 9	3
A-n60-k9	59	100	1354	2224	0.037	1354	1689.65	0.00 %	159.74	9 / 9	6
A-n61-k9	60	100	1034	2045	0.034	1037	1796.11	0.29 %	370.63	9 / 9	-1
A-n62-k8	61	100	1288	2344	0.033	1290	3867.11	0.16 %	445.29	8 / 8	-1
A-n63-k10	62	100	1314	2275	0.039	1318	3226.55	0.30 %	298.88	10 / 10	-1
A-n63-k9	62	100	1616	2659	0.039	1629	1254.34	0.80 %	183.58	9 / 9	-1
A-n64-k9	63	100	1401	2215	0.039	1431	386.78	2.14 %	-	9 / 9	-1
A-n65-k9	64	100	1174	2331	0.045	1177	1577.88	0.26 %	552.54	9 / 9	-1
A-n69-k9	68	100	1159	2463	0.068	1170	3752.35	0.95 %	1512.49	9 / 9	-1
A-n80-k10	79	100	1763	3165	0.053	1763	9145.79	0.00 %	884.94	10 / 10	20

Table 2. Results obtained for class A problems.

Problem	#Cust.	Veh.Cap.	BKS	Init.Sol.	CPU_{IS} (s)	Fin.Sol.	CPU_{FS} (s)	Gap BKS-FS (%)	$CPU_{gap\leq 2\%}$ (s)	#Veh. / Opt.#Veh.	Opt.It.
P-n16-k8	15	35	450	534	0.02	453	0.62	0.67 %	0.62	8 / 8	-1
P-n19-k2	18	160	212	267	0.156	219	0.57	3.30 %	-	2 / 2	-1
P-n20-k2	19	160	216	266	0.003	216	0.95	0.00 %	0.95	2 / 2	1
P-n21-k2	20	160	211	276	0.003	211	2.75	0.00 %	2.75	2 / 2	1
P-n22-k2	21	160	216	278	0.003	216	4.68	0.00 %	4.68	2 / 2	1
P-n22-k8	21	3000	603	687	0.009	603	1.20	0.00 %	1.20	8 / 8	1
P-n23-k8	22	40	529	642	0.016	529	2.43	0.00 %	2.43	8 / 8	8
P-n40-k5	39	140	458	773	0.014	458	39.43	0.00 %	39.43	5 / 5	1
P-n45-k5	44	150	510	827	0.015	510	111.05	0.00 %	111.05	5 / 5	1
P-n50-k7	49	150	554	1012	0.035	554	115.43	0.00 %	115.43	7 / 7	20
P-n50-k8	49	120	631	-	-	-	-	-	-	- / 8	-
P-n50-k10	49	100	696	1233	0.032	700	375.69	0.57 %	147.29	10 / 10	-1
P-n51-k10	50	80	741	1248	0.031	741	427.25	0.00 %	181.10	10 / 10	5
P-n55-k7	54	170	568	1047	0.025	568	448.47	0.00 %	224.55	7 / 7	8
P-n55-k8 (a)	54	160	588	1093	0.029	**577**	247.69	-1.87 %	247.69	**7** / 8	1
P-n55-k8 (b)	54	160	588	1116	0.031	590	1379.75	0.34 %	68.99	8 / 8	-1
P-n55-k10	54	115	694	1302	0.053	700	704.90	0.86 %	168.09	10 / 10	-1
P-n55-k15	54	70	989	-	-	-	-	-	-	- / 15	-
P-n60-k10	59	120	744	1529	0.034	744	1671.03	0.00 %	321.79	10 / 10	12
P-n60-k15	59	80	968	1761	0.052	975	1023.88	0.72 %	178.30	15 / 15	-1
P-n65-k10	64	130	792	1509	0.041	792	780.61	0.00 %	379.99	10 / 10	15
P-n70-k10	69	135	827	1586	0.055	842	3108.94	1.81 %	1293.94	10 / 10	-1
P-n76-k4	75	350	593	1062	0.068	594	12328.96	0.17 %	3740.66	4 / 4	-1
P-n76-k5	75	280	627	1177	0.042	628	10784.63	0.16 %	1167.90	5 / 5	-1
P-n101-k4	100	400	681	1124	0.156	682	82818.18	0.15 %	16563.72	4 / 4	-1

Table 3. Results obtained for class P problems. Improved solutions are marked in bold.

Problem	#Cust.	Veh.Cap.	BKS	Init.Sol.	CPU_{IS} (s)	Fin.Sol.	CPU_{FS} (s)	Gap BKS-FS (%)	$CPU_{gap\leq 2\%}$ (s)	#Veh. / Opt.#Veh.	Opt.It.
E-n101-k8	100	200	817	1628	0.199	817	40888.91	0.00 %	10021.63	8 / 8	18
E-n101-k14	100	112	1067	2106	0.106	1084	4558.74	1.59 %	3655.71	14 / 14	-1
M-n101-k10	100	200	820	1091	0.252	840	12897.92	2.44 %	-	10 / 10	-1
M-n121-k7	120	200	1034	1227	0.193	1079	39383.90	4.35 %	-	7 / 7	-1
M-n151-k12	150	200	1015	2481	0.287	1022	76933.89	0.69 %	70339.55	12 / 12	1
M-n200-k16	199	200	1371	3287	6.253	**1335**	109850.18	-2.63 %	12923.55	16 / 16	1
M-n200-k17	199	200	1275	3201	5.983	1304	195727.29	2.27 %	106251.96	17 / 17	-1
G-n262-k25	261	500	5685	14563	0.724	5710	443081.89	0.44 %	65442.80	25 / 25	1

Table 4. Results obtained for large problems (100 or more customers). Improved solutions are marked in bold.

Finally, two problems deserve special attention: M-n200-k16 and P-n55-k8. The proposed methodology is able to find a new best solution for the first and an alternative solution for the latter. A complete description of both solutions can be found in Appendix A for further revision and comparison.

For the test instance M-n200-k16, a new best solution with a value of 1335 has been obtained (Figure 4). To the best of our knowledge, only one previous feasible solution with a cost of 1371 was known (Hasle & Kloster, 2007). It was obtained by means of a VND algorithm embedded in the VRP software SPIDER. Taking into account the best known lower bound for this instance (1256.4), published in (Baldacci et al., 2008), the solution found reduces the gap between bounds from the previous value of 8.36 % to 5.89 %.

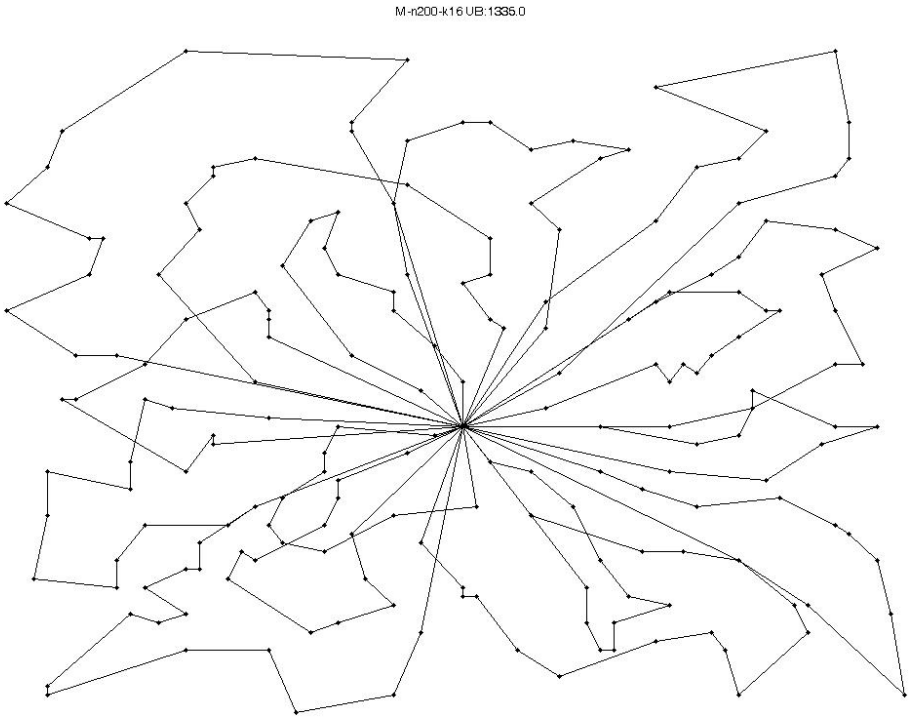


Figure 4. The best known solution so far for the test instance M-n200-k16.

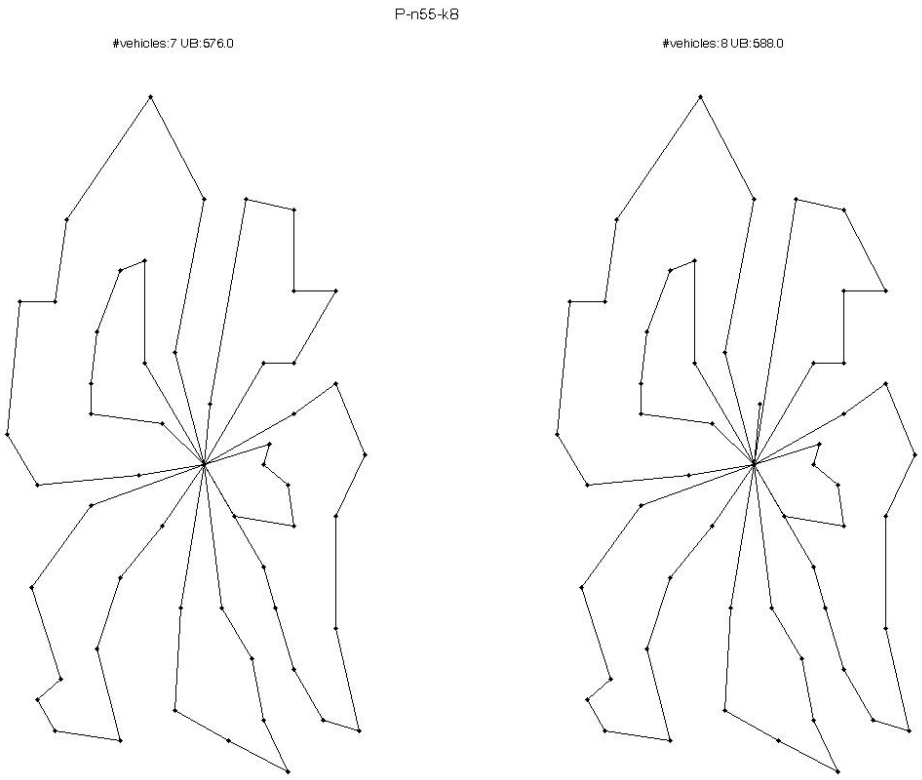


Figure 5. Alternative solution (left) to the published optimal configuration (right) for the test instance P-n55-k8. The proposed alternative solution has been obtained with a relocate/swapping configuration.

The solution found for the test instance P-n55-k8 becomes an alternative to the published optimum (Augerat et al., 1995). For this case, a solution with a value of 577 has been found by using the *swapping*/*relocate* configuration, while a value of 576 (Figure 5) has been obtained exchanging moves' priorities. Both solutions use 7 vehicles, instead of the 8 vehicles used in the published optimum. This solution has been marked as P-n55-k8 (a) in Table 3. P-n55-k8 (b) corresponds to the solution obtained when forcing the algorithm to use 8 vehicles. As far as we know, only two previous works (Alba & Dorronsoro, 2008; Altinel & Öncan, 2005) have also presented this alternative value as the best known solution for this instance, while most authors keep the original value of 588 using 8 vehicles as optimal.

In any case, optimality may not be guaranteed in the proposed solution for instance P-n55-k8. Instances P-n55-k7 and P-n55-k8 share customers' distribution and demand, but available vehicles have different capacities. Thus, both problems are critically different and the optimal value of instance P-n55-k7 can not be chosen as a reference. In fact, the solution of instance P-n55-k7 is not feasible according to P-n55-k8 vehicles' capacity. So, the proposed solution has the lowest known value for this problem, but its optimality has to be proved by means of an exact algorithm.

CONCLUSION

The present paper has presented a methodology combining Constraint Programming and Lagrangean Relaxation within a general Variable Neighbourhood Search framework. This scheme has been used to tackle the Capacitated Vehicle Routing Problem, obtaining state-of-art results comparable to other metaheuristics.

In the proposed approach, the CVRP has been decomposed into two separated subproblems. The first one is aimed to assign customers to vehicles in terms of capacity, while the second is used to optimize corresponding routes. This approach allows reducing the computation time, since problems to be solved are far less complex than the original CVRP, although still NP-hard. In fact, the allocation problem may be assimilated to the Bin Packing Problem, while routing subproblem's goal is solving a set of Travelling Salesman Problems. In both cases, two well-known paradigms aimed to solve combinatorial problems, CP and LR, have been applied obtaining good results. Thus, combining this decomposition with selected techniques provides a methodology able to get a quick initial solution to the CVRP problem, even for larger instances. Although solution's quality may be low, it may be rapidly improved by applying a local search process, such as the VND algorithm.

The proposed LR-based method permits to reduce calculation times due to its improved convergence with respect to the Subgradient Optimization classical algorithm. It also provides optimal routes when the number of customers is relatively small, as it is for all CVRP benchmark instances. Combining these characteristics with the adopted approach to the CVRP allows reducing the computation time. On the one hand, the selected decomposition makes LR only necessary to recalculate two routes at each iteration. On the other hand, the LR-based method is faster and simpler than other routing post-optimization processes, since no intra-route moves are to be defined and it is less likely to get trapped in iterative processes.

At the same time, the adopted CP approach has demonstrated to be efficient both for solving the capacity subproblem and for checking feasibility at runtime. As mentioned, a slightly different approach has been implemented to tackle feasibility checking. Unfortunately, the time required for this approach is clearly higher, due to its complexity. However, developing tailored CP algorithms to improve its efficiency constitutes an open line for future research.

It can be stated that the proposed methodology provides competitive results for most CVRP test instances. In many cases, it is able to reach best-known or optimal solutions in few iterations. Otherwise, the gap is usually low. Furthermore, the presented method has been able to provide a new best-known solution for the test instance M-n200-k16, for which only one previous feasible solution was known. An alternative solution for the test instance P-n55-k8 has also been obtained, reducing the number of vehicles and getting a lower cost.

It should be remarked that several lines for future research are open. First, different VNS schemes are to be studied, such as Variable Neighbourhood Decomposition Search, whose shaking process can be improved by embedding CP techniques. Second, heuristic methods are to be included into the neighbourhood exploration phase. Finally, the presented methodology is to be adapted to different VRP variants, especially those including time windows or pick-up and delivery side constraints.

REFERENCES

Alba, E., & Dorronsoro, B. (2008). A hybrid cellular genetic algorithm for the capacitated vehicle routing problem. In Abraham, A., Grosan, C., & Pedrycz, W. (Eds.), *Engineering Evolutionary Intelligent Systems (Studies in Computational Intelligence, 82)* (pp. 379-422). Berlin, Germany: Springer-Verlag.

Altinel, I., & Öncan, T. (2005). A new enhancement of the Clarke and Wright savings heuristic for the capacitated vehicle routing problem. *Journal of the Operational Research Society*, 56, 954-961.

Apt, K., & Wallace, M.G. (2007). *Constraint Logic Programming using ECLiPSe*. Cambridge, UK: Cambridge University Press.

Augerat, P., Belenguer, J.M., Benavent, E., Corberán, A., Naddef, D., & Rinaldi, G. (1995). *Computational Results with a Branch and Cut Code for the Capacitated Vehicle Routing Problem* (Research Report No. 949-M). Grenoble, France: Université Joseph Fourier.

Baldacci, R., Christofides, N., & Mingozzi, A. (2008). An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2), 351-385.

Beldiceanu, N., Carlsson, M., & Rampon, J.-X. (2005). *Global Constraint Catalog* (Tech. Rep. No. 2005:08). Kista, Sweden: Swedish Institute of Computer Science (SICS).

Bellman, R. (1962). Dynamic Programming Treatment of the Travelling Salesman Problem. *Journal of ACM*, 1, 61-63.

Bessiere, C. (2006). Constraint propagation. In Rossi, F., van Beek, P., & Walsh, T. (Eds.), *Handbook of Constraint Programming* (pp. 29-83). Amsterdam, The Netherlands: Elsevier.

Boschetti, M., & Maniezzo, V. (2009). Benders decomposition, Lagrangean relaxation and metaheuristic design. *Journal of Heuristics*, 15, 283-312.

Branchandcut.org (2003). Vehicle Routing Data Sets. Downloaded November 13, 2009, from http://branchandcut.org/VRP/data/

Bräysy, O. (2003). A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS Journal on Computing*, 15(4), 347–368.

Bräysy, O., & Gendreau, M. (2005). Vehicle routing problems with time windows, part II: metaheuristics. *Transportation Science*, 39(1), 119-139.

Cordeau, J.-F., & Laporte, G. (2004). Tabu search heuristics for the vehicle routing problem. In Rego, C., & Alidaee, B. (Eds.), *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search* (pp. 145-163). Boston, MA: Kluwer Academic.

Cordeau, J.-F., Laporte, G., Savelsbergh, M.W.P., & Vigo, D. (2007). Vehicle routing. In Barnhart, C., & Laporte, G. (Eds.), *Handbook in Operations Research and Management Science, Vol. 14* (pp. 367-428). Amsterdam, The Netherlands: Elsevier.

Dantzig, G.B., & Ramser, J.H. (1959). The truck dispatching problem. *Management Science*, 6, 80-91.

Fisher, M.L. (1981). The Lagrangean relaxation method for solving integer programming problems. *Management Science*, 27**,** 1–18

Guignard, M. (2003). Lagrangean Relaxation. *Sociedad de Estadística e Investigación Operativa, Top*, 11(2), 151-228.

Hansen, P. & Mladenovic, N. (2003). *A tutorial on variable neighborhood search* (Tech. Rep. No. G-2003-46). Montreal, Canada: Les Cahiers du GERAD, HEC Montreal and GERAD.

Hasle, G., & Kloster, O. (2007). Industrial vehicle routing. In Hasle, G., Lie, K.-A., & Quak, E. (Eds.), *Geometric Modelling, Numerical Simulation, and Optimization* (pp. 397-435). Berlin, Germany: Springer-Verlag.

Held, M., & Karp, R.M. (1971). The Travelling Salesman Problem and Minimum Spanning Trees: Part II. *Mathematical Programming*, 1, 6-25.

Held, M., Wolfe, P., & Crowder, H. (1974). Validation of Subgradient Optimization. *Mathematical Programming*, 6, 62-88.

Laporte, G. (1992). The traveling salesman problem: an overview of exact and approximate algorithms. *European Journal of Operational Research*, 59, 231–247.

Mladenovic, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11), 1097-1100.

Reeves, C. (2003). Genetic algorithms. In Glover, F., & Kochenberger, G.A. (Eds.), *Handbook of Metaheuristics* (pp. 55-82). Boston, MA: Kluwer Academic.

Reinelt, G. (1994). *The Travelling Salesman: Computational Solutions for TSP Applications*. Berlin: Springer-Verlag.

Reinelt, G. (2008). TSPLIB. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/. Accessed January 28, 2010.

Rossi, F., van Beek, P., & Walsh, T. (Eds.). (2006). *Handbook of Constraint Programming*. Amsterdam, The Netherlands: Elsevier.

Rousseau, L.M., Gendreau, M., & Pesant, G. (2002). Using constraint-based operators to solve the vehicle routing problem with time windows. *Journal of Heuristics*, 8, 43-58.

Savelsbergh, M.W.P. (1988). Computer aided routing. *Centrum voor Wiskunde en Informatica*.

Savelsbergh, M.W.P. (1985). Local search in routing problems with time windows. *Annals of Operations Research*, 4, 285-305.

Wolsey, L.A. (1998). *Integer programming*. New York: John Wiley & Sons.

Zamani, R., & Lau, S.K. (2010). Embedding learning capability in Lagrangean relaxation: An application to the travelling salesman problem. *European Journal of Operational Research*, 201, 82-88.

APPENDIX A

This appendix provides information about new best solutions found for problems M-n200-k16 and P-n55-k8. In both cases, vertex 1 corresponds to the depot.

M-n200-k16

Cost: 1335
Route #1: 1, 119, 61, 6, 46, 126, 115, 83, 8, 195, 107, 154, 1
Route #2: 1, 113, 184, 95, 96, 38, 99, 101, 43, 173, 145, 88, 118, 1
Route #3: 1, 54, 106, 181, 22, 73, 198, 75, 134, 23, 172, 74, 1
Route #4: 1, 28, 168, 128, 191, 89, 149, 160, 63, 183, 53, 147, 1
Route #5: 1, 155, 139, 110, 178, 151, 69, 164, 25, 135, 55, 196, 1
Route #6: 1, 13, 81, 122, 30, 170, 79, 35, 165, 121, 82, 34, 103, 1
Route #7: 1, 19, 49, 124, 20, 108, 176, 12, 109, 71, 102, 163, 70, 133, 1
Route #8: 1, 5, 140, 40, 24, 187, 57, 76, 42, 146, 116, 179, 3, 138, 1
Route #9: 1, 59, 153, 14, 98, 93, 152, 60, 97, 7, 148, 90, 157, 1
Route #10: 1, 105, 94, 62, 17, 142, 87, 114, 18, 174, 85, 200, 84, 167, 1
Route #11: 1, 9, 175, 47, 125, 169, 48, 37, 144, 50, 65, 182, 64, 127, 190, 1
Route #12: 1, 2, 52, 104, 162, 72, 67, 66, 137, 36, 136, 10, 112, 1
Route #13: 1, 29, 77, 185, 197, 117, 78, 4, 159, 130, 80, 186, 158, 51, 1
Route #14: 1, 58, 16, 44, 143, 15, 39, 141, 45, 120, 193, 192, 92, 194, 86, 100, 1
Route #15: 1, 27, 150, 180, 131, 166, 56, 26, 171, 68, 188, 156, 111, 199, 41, 1
Route #16: 1, 32, 11, 91, 33, 132, 161, 129, 189, 21, 31, 123, 177, 1

P-n55-k8

Cost: 576
Route #1: 1, 8, 36, 15, 54, 12, 39, 27, 1
Route #2: 1, 3, 29, 22, 37, 48, 49, 31, 1
Route #3: 1, 13, 11, 32, 26, 51, 19, 25, 50, 52, 1
Route #4: 1, 5, 28, 53, 35, 47, 1
Route #5: 1, 7, 34, 2, 23, 43, 42, 44, 24, 17, 1
Route #6: 1, 9, 20, 55, 14, 16, 21, 38, 6, 30, 46, 1
Route #7: 1, 18, 4, 45, 33, 10, 40, 41, 1