Combining Constraint Programming, Lagrangian Relaxation and Probabilistic Algorithms to solve the Vehicle Routing Problem

Daniel Guimarans[1], Rosa Herrero[1], Daniel Riera[2], Angel A. Juan[2], and Juan José Ramos[1]

[1] Departament de Telecomunicació i Enginyeria de Sistemes
Universitat Autònoma de Barcelona (UAB)
Barcelona, Spain
daniel.guimarans,rosa.herrero,juanjose.ramos@uab.cat

[2] Estudis d'Informàtica, Multimèdia i Telecomunicació
Universitat Oberta de Catalunya (UOC)
Barcelona, Spain
drierat,ajuanp@uoc.edu

Abstract

This paper presents a hybrid approach that aims at solving the Capacitated Vehicle Routing Problem (CVRP) by means of combining Constraint Programming (CP) with Lagrangian Relaxation (LR) and Probabilistic Algorithms. After introducing the CVRP and reviewing the main literature in this area, the paper proposes the use of a multi-start hybrid Variable Neighbourhood Search (VNS) algorithm. This algorithm uses a randomised version of the classical Clarke and Wright savings heuristic to generate a starting solution to a given CVRP. This starting solution is then improved through a local search process which combines: (a) LR to optimise each individual route, and (b) CP to quickly verify the feasibility of new proposed solutions. Some results on well-known CVRP benchmarks are analysed and discussed.

1 Introduction

Road transportation is nowadays the predominant way of transporting goods in many parts of the world. Direct costs associated with this type of transportation have experienced a significant increase since 2000.

There is a need for developing efficient models and methods that support decision-making processes in the road transportation arena so that optimal (or quasi-optimal) strategies can be chosen. This necessity for optimising road transportation, which affects both the public and private sectors, constitutes a major challenge for most industrialised regions.

The class of logistic problems dealing with the issues presented in the previous paragraph are usually known as the Vehicle Routing Problem (VRP). This is among the most popular research areas in combinatorial optimisation. It was first defined by Dantzig and Ramser in 1959 [6], and several

Proceedings of the 17th International RCRA workshop (RCRA 2010):
Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion
Bologna, Italy, June 10–11, 2010

variants of the basic problem have been proposed and studied later. The most basic VRP is the CVRP, which assumes a fleet of vehicles of homogeneous capacity housed in a single depot. It is a generalisation of the Travelling Salesman Problem (TSP) and is therefore NP-hard [21]. For such problems, even finding an initial solution is NP-hard and often requires a high computational effort.

For this paper, the CVRP has been chosen. It is mainly because there are a huge amount of models, techniques, benchmarks and research — in general — done before (See Section 3). Hence, results given by the presented methodology can be compared with a number of previous ones.

The paper presents a general VNS structure whose local search process is supported by Constraint Programming (CP) and lagrangian Relaxation (LR). A probabilistic Clarke and Wright Savings (CWS) constructive method is used to generate initial solutions.

The remainder of this article is structured as follows. Next sections provide a general overview of CVRP formulation and a few words on previous work.An introduction to the CVRP mathematical model and the technologies used in this research are presented in sections 4 and 5 respectively. Section 6 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.

2 The Capacitated Vehicle Routing Problem

The CVRP is defined over a complete graph $G\{I, E\}$ connecting the vertex set $I = \{1, 2, \ldots, n\}$ — clients to be served and the depot — through a set of edges $E = \{(i, j)|i, j \in I\}$ — roads, streets, etc. Edges $e_{ij} \in E$ have an associated a travel cost c_{ij}, which is supposed to be the lowest cost route connecting node i to node j. Usually, to simplify it (not the complexity but the problem definition), edges cost is assumed as symmetric ($c_{ij} = c_{ji}$). Each vertex $i \in I\backslash\{1\}$ has a nonnegative demand q_i, while, vertex 1 corresponds to the depot, which does not have an associated demand.

A fixed fleet of m identical vehicles, each one with 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 starting and finishing vertex (i.e. the depot).

3 Previous work on the Capacitated Vehicle Routing Problem

CWS constructive algorithm [4] is probably the most cited heuristic to solve the CVRP. The CWS is an iterative method that starts out by considering an initial dummy solution in which each customer is served by a dedicated vehicle. Next, the algorithm initiates an iterative process for merging some of the routes in the initial solution.

The CWS algorithm usually provides relatively good solutions, especially for small and medium-size problems. Many variants and improvements of the CWS have been proposed in the literature [15].

Using constructive heuristics as a basis, metaheuristics became popular for the VRP during the nineties. Some early examples are the Tabu Route method by Gendreau et al. [11] or the Boneroute method of Tarantilis and Kiranoudis [23]. Tabu search algorithms, like the one proposed by Toth and Vigo [24] is among the most cited metaheuristics. Genetic algorithms have also played a major role in the development of effective approaches for the VRP (e.g. [18]). Another important approach to the VRP is given by the Greedy Randomised Adaptive Search Procedure (GRASP) [8].

Monte Carlo Simulation (MCS) can be defined as a set of techniques that make use of random numbers and statistical distributions to solve certain stochastic and deterministic problems [16]. MCS has proved to be extremely useful for obtaining numerical solutions to complex problems that cannot be efficiently solved by analytical approaches. Buxey [3], in 1979, was probably the first author to combine MCS with the CWS algorithm to develop a procedure for the CVRP. This method was revisited by Faulin and Juan [7], who introduced an entropy function to guide the random selection of nodes.

Another way to face the VRP has been the use of complete methods, which ensure not only to find the solution but also, to prove its optimality. The main drawback of these techniques is that they may only deal with small instances, up to 100 customers [5]. Numerous heuristics and metaheuristics (like those presented above) have also been studied for different VRP variants. In most cases, these methods may solve larger instances but loosing optimality guarantees. Among metaheuristics, Variable Neighbourhood Search (VNS), introduced for the first time by Mladenovic and Hansen [17], is a quite a recent method with far less application examples in VRP research. However, interesting results have been obtained even applying the simplest VNS algorithm (e.g. [13]). For this reason, VNS has been selected as the general framework where to embed CP and 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.

4 Mathematical Model of the CVRP

In the proposed model, the CVRP has been divided into two subproblems, concerning customers' allocation and routing optimisation separately. The first is aimed to assign customers to the minimum number of required vehicles fulfilling capacity limitations. The latter is used to solve each independent route to optimality, giving the best solution for a particular allocation.

4.1 Capacity problem

The proposed customers' allocation subproblem uses two lists of variables. A list R of size n, with integer domains $R_i \in [1..m]|i = 1, ..., n$, indicates which vehicle is serving the i^{th} customer. Given a vehicle v, Q_v is a list of m variables with real domain $Q_v \in [0..Q]$ used to trace the cumulative capacity at each one of the m routes. Therefore, capacity constraints are enforced through domains definition since Q_v cannot get higher values than the maximum capacity Q.

A set of $m \times n$ binary variables B has been introduced to relate R and Q_v values. For each vehicle $v \in V$, a list of n binary variables $B_{vi} \mid i \in I$ 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.

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

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

Cumulative capacities can be traced simply by using Eq.2.

$$Q_v = \sum_{i \in I} B_{vi} q_i \ \forall v \in V \tag{2}$$

4.2 Routing Problem

The routing problem, tackled for each vehicle separately, can be viewed as a TSP instance. For each vehicle v, the related TSP can be considered as a complete undirected graph $G = (I_v, E_v)$, connecting assigned customers $I_v = \{i \in I | R_i = v\}$ through a set of undirected edges $E_v = \{(i, j) \in E | i, j \in I_v\}$. 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.

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. That is $x_e = 1$ if customer j is visited immediately after i; otherwise $x_e = 0$. Thus, the formulation for the TSP problem is as follows:

$$\min \sum_{e \in E_v} c_e x_e \tag{3}$$

subject to

$$\sum_{e \in \delta(i)} x_e = 2 \ , \qquad \forall i \in I_v \tag{4}$$

$$\sum_{e \in E_v(S)} x_e \leq \mid S \mid -1 \ , \qquad \forall S \subset I_v \ , \ \mid S \mid \leq \frac{1}{2} \mid I_v \mid \tag{5}$$

where $\delta(i) = \{e \in E_v : \exists j \in I_v,\ e = (i,j)\ or\ (j,i)\}$ represents the set of arcs whose starting or ending node is i; $E_v(S) = \{e_{ij} \in E_v : i,j \in S\}$ represents the set of arcs whose nodes is in the subset S of vertices; $n_v = |I_v|$; and c_e is the associated cost to the undirected edge $e_{ij}(e_{ji})$.

Constraint (4) states that every node $i \in I_v$ must be visited once, that is, every customer must have two incident edges. Subtour elimination constraint (5) states that the tour must be a Hamiltonian path, so it cannot have any subcycle. 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.

5 Technologies Used

5.1 Probabilistic Clarke and Wright Savings Algorithm

CWS is an iterative method that starts out by considering an initial dummy solution in which each customer is served by a dedicated vehicle. Next, the algorithm initiates an iterative process for merging some of the routes in the initial solution. Merging routes can improve the expensive initial solution so that a unique vehicle serves the nodes of the merged route. The merging criterion is based upon the concept of savings. Roughly speaking, given a pair of nodes to be served, a savings value can be assigned to the edge connecting these two nodes. This savings value is given by the reduction in the total cost function due to serving both nodes with the same vehicle instead of using a dedicated vehicle to serve each node.This way, the algorithm constructs a list of savings, one for each possible edge connecting two demanding nodes. At each iteration of the merging process, the edge with the largest possible savings is selected from the list as far as the following conditions are satisfied: (a) the nodes defining the edge are adjacent to the depot, and (b) the two corresponding routes can be feasibly merged — i.e. the vehicle capacity is not exceeded.

The selection process should be done without introducing too many parameters in the algorithm. To do it, different geometric statistical distributions during the randomised CWS solution-construction process are em-

ployed: every time a new edge is selected from the list of available edges, a value α is randomly selected from a uniform distribution in (a, b), where $0 < a \leq b < 1$. This parameter α defines the specific geometric distribution that will be used to assign exponentially diminishing probabilities to each eligible edge according to its position inside the sorted savings list. That way, edges with higher savings values are always more likely to be selected from the list, but the exact probabilities assigned are variable and they depend upon the concrete distribution selected at each step.

5.2 Constraint Programming

CP 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 optimisation, or even tailored search methods for specific problems. Rossi et al. present a complete overview of CP modelling techniques, algorithms, tools and applications [10].

5.3 Lagrangian Relaxation

LR is a well-known method to solve large-scale combinatorial optimisation 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 [9].

LR exploits the structure of the problem, so it reduces considerably problem's complexity. However, it is often a major issue to find optimal Lagrangian multipliers. The most commonly used algorithm is the Subgradient Optimisation (SO). The main difficulty of this algorithm lays on choosing a correct step-size λ_k in order to ensure algorithm's convergence [19].

Therefore, the proposed method combines the SO algorithm with a heuristic to obtain a feasible solution from a dual solution. It can get a better upper bound UB, so it improves the convergence on the optimal solution departing from an initial UB obtained with a Nearest Neighbour Heuristic. 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.

5.4 Variable Neighbourhood search

A general VNS, as explained in [12], is a recent metaheuristic which exploits systematically the idea of neighborhood change, both in descent to local minima and in escape from the valleys which contain them. A diversification process (*shaking*) ensures that different regions of the search space are

explored. Every time the *shaking* process generates a new solution, it is improved by means of a Variable Neighbourhood Descent (VND) method.

The VNS starts from an initial solution x. A new point is generated at random from the k^{th} neighbourhood $N_k(x)$ of x in order to diversify the search. 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 shaking process is repeated.

The local search process for each neighbourhood $N_l(x')$ performs an exhaustive exploration with a best-accept strategy. Finally, the best neighbour x'' is chosen in terms of its solution value $f(x'') = \sum_{v \in V} UB_v$. If its value is lower than the original $f(x')$, the solution is updated and neighbourhoods exploration is restarted.

When the VND process reaches a local optimum, no solution improvement may be found according to defined neighbourhoods. If this local optimum is better than the incumbent, it is accepted as the current solution $x \leftarrow x''$ and the search is restarted from the first shaking neighbourhood. Otherwise, the algorithm keeps x as the best solution found so far and continues exploring the next neighbourhood. If there are no remaining neighbourhoods to be explored, the process is restarted until the stopping condition is met.

6 The Proposed Methodology

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 VNS algorithm, known as VND, has provided promising results when solving different VRP variants. In the proposed approach, a general VNS framework has been chosen to embed selected paradigms. Within the VNS general framework, CP and LR are used in different processes. During algorithm's initialisation, a Probabilistic CWS method has been used to find an initial feasible solution [14].

CP is 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 initialisation, diversification or local search processes. Using LR allows reducing the computation time when compared to other routing post-optimisation 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, allows reducing algorithm's definition and complexity.

6.1 Lagrangian Relaxation Method Applied

The LR-based method is used within the local search process to solve the routing problem to optimality. It can be considered a specification of the Lagrangian Metaheuristic presented on Boschetti and Maniezzo [2]. It uses the SO algorithm combined with a heuristic. Aiming to improve algorithm's convergence to the optimum, 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 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 ($\gamma^k < 10^{-15}$). The proposed LR-based method algorithm is shown in Table 1.

0	Initialisation
1	Initialise parameters $u^0 = 0; \delta_0 = 2; \rho = 0.95; \alpha_L = 1/3$
2	Obtain an UB applying Nearest Neighbour Heuristic
3	Initialise $\overline{L} = L(u^0) + \alpha_L(UB - L(u^0))$
4	Iteration k
5	Solve the Lagrangian function $L(u^k)$
6	Check the subgradient $\gamma_i^k = 2 - \sum_{e \in \delta(i)} x_e$
7	if $\parallel \gamma^k \parallel^2 = 0$ then Optimal solution is found $\Rightarrow$ EXIT
8	if $\parallel \gamma^k \parallel^2 < \xi$ then apply a heuristic to improve the UB
9	Check the parameter $\overline{L}$
10	Calculate the step-size $\lambda_k = \delta_k \frac{\overline{L} - L(u^k)}{\|\gamma_k\|^2}$
11	Update the multiplier $u^{k+1} = u^k + \lambda_k \gamma^k$
12	$k \leftarrow k + 1$

Table 1: The Proposed LR-based Method Algorithm

The proposed heuristic to improve the UB is applied when the solution is nearly a route. That is, if it satisfies $\parallel \gamma \parallel^2 < \xi$ (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 $e = (i, j)$ where j has some extra edges for an edge $e = (i, l)$ 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:

$$\{i, j, l\} = argmin\Big\{c_{il} - c_{ij} \mid \gamma_j < 0,\ \gamma_l > 0,\ (i, j) \in T,\ (i, l) \in E \setminus T\Big\} \quad (6)$$

The parameter ξ depends on the number of variables. A good estimation of ξ value would avoid increasing the computation time excessively. First,

its value may be large, for instance $n_v/2$, but it should be updated whenever a feasible solution is found according to $\xi = \| \gamma^k \|^2$. 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 $LB \leq \overline{L} \leq UB$, is proposed. By definition, this parameter corresponds to a better estimation of L^* than those obtained for LB and UB. The calculation of the step-size turns into:

$$\lambda_k = \delta_k \frac{\overline{L} - L(u^k)}{\| \gamma_k \|^2}$$

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 initialised $\overline{L} = L(u^0) + \alpha_L(UB - L(u^0))$ with $0 < \alpha_L < 1$; if $L(u^k) > \overline{L}$, it is updated $\overline{L} = L(u^k) + \alpha_L(UB - L(u^k))$, and if $\overline{L} > UB$, then $\overline{L} = UB$.

Finally, the parameter δ_k is initialised to the value 2 and is updated as Zamani and Lau [25] 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 $\delta_{k+1} = \delta_k \frac{3-\rho}{2}$.

6.2 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 [22] have been defined so they can be used within diversification and local search processes: (a) *Relocate* moves a customer from one route to a different one, (b) *Swapping* exchanges two customers belonging to two different routes, (c) *Chain* is a specialisation of 3-opt that swaps sections of two contiguous customers from two different routes, and (d) *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 optimisation process is needed. Usually, a post-optimisation method based on intra-route moves is applied to improve each single route quality [20].

6.3 Variable Neighbourhood Search Framework

A general VNS framework, as explained in Hansen and Mladenovic [12], has been implemented embedding the described methods. A simplified scheme of the method is presented in Table 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.

0	Initialisation. Select the set of neighbourhood structures N_k, for $k = 1, \ldots, k_{max}$, that will be used in the shaking phase, and the set of neighbourhood structures N_l for $l = 1, \ldots, l_{max}$ 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 $k \leftarrow k + 1$;

Table 2: Variable Neighbourhood Search Algorithm

In step 0, neighbourhood structures to be used within shaking (N_k) and local search (N_l) processes are selected. In this case, all four described moves have been selected to be used in both neighbourhoods. Furthermore, a randomised CWS algorithm is used in order to find a good initial solution for the search.

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. 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.

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 $f(x'') = \sum_{v=1}^{m} UB_v$.

7 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 [1]. All tests have been performed on a non-dedicated server with an Intel i5 processor at 2.66GHz and 16GB RAM. In general, five to seven processes were launched in parallel to solve different problems.

A total of 93 CVRP benchmark instances from www.branchandcut.org have been used to test the efficiency of the proposed approach. Only those instances whose distance is defined as Euclidean or Geographic have been selected, in order to ensure triangular inequality's fulfilment. Table 3 shows the total number of problems chosen from each class, as well as for how many the best known solution has been reached. This table also shows the average gap (% Aver. Gap) from best published values for those problems where the best known solution could not be reached after 40 iterations. A low deviation, under 1 %, is observed for most problem sets, comparable to results obtained by means of other metaheuristics.

Class	Problems	Opt.	No opt.	% Aver.Gap
A	27	15 (55.6 %)	12 (44.4 %)	0.74
B	23	12 (52.2 %)	11 (47.8 %)	0.61
E	11	4 (36.4 %)	7 (63.6 %)	0.93
F	3	1 (33.3 %)	2 (66.7 %)	1.02
G	1	1 (100.0 %)	0 (0.0 %)	0.00
M	5	1 (20.0 %)	4 (80.0 %)	2.14
P	21	11 (52.4 %)	10 (47.6 %)	0.28
TSPLib	2	1 (50.0 %)	1 (50.0 %)	1.33
	93	46 (49.5 %)	47 (50.5 %)	0.72

Table 3: Summary of results obtained with the proposed methodology.

The use of LR ensures the partial optimality of all solutions from the

routing perspective. The reason is that the proposed approach can optimally solve all TSP instances, due to the number of associated customers is always low. LB and UB converge rapidly, keeping their gap between 0 and 10^{-10} and guaranteeing so the solution optimality. In addition, LR solves all routes in negligible times. Thus, LR has demonstrated to be an efficient alternative for intra-route optimisation processes.

The initial solution may be obtained by means of different methods. Table 4 presents a comparative of results for some problems of class A using different techniques to get an initial solution. First, it may be obtained by solving separately capacity and routing problems, using the proposed problem's decomposition in a CP/LR scheme (2). This approach is able to provide a low-quality quick solution, since both subproblems are easily solved but variables are unlinked. This solution may be highly improved applying a VND method, providing an initial solution whose value is usually close to the final result (1). The initial solution may also be obtained by means of the presented RCWS algorithm (3). This algorithm provides a good initial solution in negligible times, but the maximum number of available vehicles might not be always respected. For this reason, it may be forced to execute iteratively until a solution fulfilling this requirement is reached (4).

It can be stated from results presented in Table 4 that RCWS is clearly faster than the other methods to get a good initial solution. Even when it is forced to iterate to reach a given number of vehicles, it may provide a solution quickly. It is remarkable that this algorithm eventually finds a better initial solution than the CP/LR + VND in a much lower time, becoming so a better alternative. For small problems, the RCWS + VNS scheme outperforms the CP/LR + VNS method, reaching best known values in lower times. On the other hand, times are comparable for larger problems. The reason might be found in the use of exhaustive local search for all defined moves. This fact also justifies the time spent on calculating the initial solution when the CP/LR + VND method is used. For small problems, feasible points around a given solution may be usually low, while the solution space may grow dramatically as the problem size increase, and so it does the time needed to explore it. Probably, the use of heuristics within local search processes would reduce the total time needed to solve problems significantly.

Using RCWS to get an initial solution allows solving to the best known value 15 of 27 problems from class A. The same number of problems are solved departing from an initial solution obtained by means of the CP/LR scheme without VND. Instead, CP/LR + VND allows solving only 13 problems. This method uses all four defined moves to reach a minimum. As they are also used in the VNS algorithm, the initial solution is a local optimum for all moves defined in its local search processes. Thus, a more thorough *shaking* would be mandatory to get a better algorithm's performance and avoid getting trapped in local minima. Moreover, the algorithm fails to successfully solve the same problems, regardless which method is used to get

the initial solution. Again, a poor *shaking* might be the main reason for this behaviour. Table 4 also shows the gap between the final solution and the best known value, when it is not reached. It can be observed it is usually lower when a good initial solution is provided.

RCWS has been used to get the initial solution for all tested instances due to its performance. Table 5 shows results obtained using RCWS combined with VNS for some representative instances of different sizes.

This methodology performs similarly both for small and large instances. 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. Thus, its applicability is not restricted. Nevertheless, the time required to solve all instances is prohibitive when compared to state-of-art methodologies. For this reason, some research needs to be addressed to improving local search processes in order to get competitive results.

Notice that results for the largest selected test instance G-n262-k25 is 1.95% lower than its best value published in [13]. This reduces the gap for this problem from 10.92% to 9.15%. The solution found for the test instance P-n55-k8 becomes an alternative to its best known value. since a solution with cost 583 using only 7 vehicles has been found, instead of the original of 588 with 8 vehicles.

8 Conclusions and Future Work

Use of multiple agents or algorithm's instances — each of them initiated with a different random-generator seed or a different value for the statistical distribution parameter — can significantly diminish computational times to obtain pseudo-optimal solutions. Larger problems might need either more computation resources or larger computational times to be solved.

In such scenarios, one possible approach to allow for real-time decision-making could be to employ modern GPUs. This is one of the most promising research lines we are currently exploring. However, use of GPUs requires advanced algorithmic and programming skills, since developing efficient code in CUDA is not a trivial task and, in the case of our algorithm, some problems related to memory management must be addressed yet before obtaining competitive results. An alternative approach is to use a distributed problem-solving approach. This way, pseudo-optimal solutions for large and complex real-life problems might be obtained in nearly real time at an inexpensive monetary cost.

In effect, a standard small and medium enterprises (SMEs) use a large number of commodity computers distributed among their different departments and/or facilities. Most of these personal computers offer more computing capabilities than required to complete their daily activities (i.e. text processors, spreadsheets, e-mail, etc.). Moreover, they happen to be un-

derutilized during nightly hours. Our proposal gathers the spare resources from each computer and aggregates them into a computational environment where hundreds or even thousands of instances of our algorithm can be run simultaneously. These cloned agents can either use a global collaborative memory or several local collaborative memories, depending on the characteristics of the distributed system.

Acknowledgments

This work was supported in part by the IN3-UOC Knowledge Community Program under grant HAROSA09.

References

[1] K. Apt and M. Wallace. *Constraint Logic Programming using ECLiPSe*. Cambridge University Press, 2007.

[2] M. Boschetti and V. Maniezzo. Benders decomposition, lagrangean relaxation and metaheuristic design. *Journal of Heuristics*, 15:283–312, 2009.

[3] G. Buxey. The vehicle scheduling problem and monte carlo simulation. *Journal of Operational Research Society*, 30:563–573, 1979.

[4] G. Clarke and J. Wright. Scheduling of vehicles from a central depot to a number of delivering points. *Operations Research*, 12:568–581, 1964.

[5] J. Cordeau, G. Laporte, M. Savelsbergh, and D. Vigo. Vehicle routing. In C. Barnhart and G. Laporte, editors, *Handbook in Operations Research and Management Science*, volume 14, pages 367–428. Elsevier, Amsterdam, The Netherlands, 2007.

[6] G. Dantzig and J. Ramser. The truck dispatching problem. *Management Science*, 6:80–91, 1959.

[7] J. Faulin and A. A. Juan. The algacea-1 method for the capacitated vehicle routing problem. *International Transactions in Operational Research*, 15:1–23, 2008.

[8] P. Festa and M. Resende. An annotated bibliography of grasp- part i: Algorithms. *International Transactions in Operational Research*, 16:1–24, 2009.

[9] M. Fisher. The lagrangean relaxation method for solving integer programming problems. *Management Science*, 27:1–18, 1981.

[10] E. Freuder, C. Bessiere, P. van Beek, H. Hoos, E. Tsang, W. van Hoeve, I. Katriel, R. Dechter, D. Cohen, P. Jeavons, P. Messeguer, F. Rossi, T. Schiex, I. Gent, K. Petrie, J. Puget, S. B.M, K. Marriot, P. Stuckey, M. Wallace, T. Fruhwirth, L. Michel, C. Schulte, M. Carlsson, J. Hooker, F. Benhamou, L. Granvilliers, C. Gervet, C. Gomes, T. Walsh, M. Koubarakis, B. Faltings, K. Brown, I. Miguel, P. Baptiste, P. Laborie, C. Le Pape, W. Nuijten, P. Kilby, P. Shaw, U. Junker, H. Simonis, R. Backofen, and D. Gilbert. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, pages 29–83. Elsevier, 2006.

[11] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40:1276–1290, 1994.

[12] P. Hansen and N. Mladenovic. A tutorial on variable neighborhood search. Technical Report G-2003-46, Groupe d'Études et de Recherche en Analyse des Décisions (GERAD), Montreal, Canada, 2003.

[13] G. Hasle and O. Kloster. *Industrial vehicle routing*, pages 397–435. Springer-Verlag, Berlin, Germany, 2007.

[14] A. Juan, J. Faulin, J. Jorba, D. Riera, D. Masip, and B. Barrios. On the use of monte carlo simulation, cache and splitting techniques to improve the clarke and wright savings heuristics. *Journal of the Operational Research Society*, To be published.

[15] G. Laporte. What you should know about the vehicle routing problem. *Naval Research Logistics*, 54:811–819, 2007.

[16] A. Law. *Simulation Modeling & Analysis*. McGraw-Hill, 2007.

[17] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[18] Y. Nagata. Edge assembly crossover for the capacitated vehicle routing problem. *Lecture Notes in Computer Science*, (4446), 2007.

[19] G. Reinelt. The traveling salesman: computational solutions for tsp applications. *Lecture notes in computer science*, 840, 1994.

[20] L. Rousseau, M. Gendreau, and G. Pesant. Using constraint-based operators to solve the vehicle routing problem with time windows. *Journal of Heuristics*, 8:43–58, 2002.

[21] M. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4:285–305, 1985.

[22] M. Savelsbergh. Computer aided routing. Technical report, Centrum voor Wiskunde en Informatica, 1988.

[23] C. Tarantilis and C. Kiranoudis. Boneroute: an adaptative memory-based method for effective fleet management. *Annals of Operations Research*, 115:227–241, 2002.

[24] P. Toth and D. Vigo. The granular tabu search and its application to the vehicle routing problem. *INFORMS Journal on Computing*, 15:333–346, 2003.

[25] R. Zamani and S. Lau. Embedding learning capability in lagrangean relaxation: An application to the travelling salesman problem. *European Journal of Operational Research*, 201(1):82–88, 2010.

Problem	BKS	Init.Sol.				CPU (s)			
		(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
A-n32-k5	784	989	1243	865	849	1.75	0.06	0.01	0.01
A-n33-k5	661	734	1185	741	693	1.71	0.01	0.00	0.01
A-n33-k6	742	792	1289	768	809	1.84	0.01	0.01	0.01
A-n45-k6	944	1006	1826	1018	988	6.15	0.02	0.00	0.02
A-n46-k7	914	986	1711	959	990	7.61	0.02	0.00	0.00
A-n54-k7	1167	1178	1883	1191	1187	20.17	0.03	0.00	0.00
A-n55-k9	1073	1084	2074	1112	1150	19.62	0.03	0.00	0.00
A-n65-k9	1174	1315	2331	1296	1263	26.77	0.04	0.01	0.01
A-n69-k9	1159	1194	2463	1258	1200	71.32	0.04	0.01	0.00
A-n80-k10	1763	1963	3165	1854	1888	89.32	0.05	0.01	0.01

Problem	BKS	Fin. Sol.				CPU (s)			
		(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
A-n32-k5	784	784	784	784	784	13.12	24.50	33.05	58.10
A-n33-k5	661	661	661	661	661	24.52	16.24	5.89	18.76
A-n33-k6	742	742	742	742	743	160.10	161.32	45.58	19.94
A-n45-k6	944	963	976	949	948	169.23	376.49	282.60	24.21
A-n46-k7	914	914	914	914	914	372.14	93.30	63.98	78.47
A-n54-k7	1167	1172	1174	1167	1167	93.31	140.43	84.13	1130.79
A-n55-k9	1073	1073	1073	1073	1073	731.67	262.96	105.03	351.51
A-n65-k9	1174	1178	1182	1178	1184	924.21	1761.44	1606.04	1824.30
A-n69-k9	1159	1167	1167	1173	1164	798.56	1064.37	3394.83	2351.63
A-n80-k10	1763	1783	1787	1778	1779	3032.76	6458.92	3429.04	5878.78

Problem	BKS	Gap BKS-FS (%)				# it			
		(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
A-n32-k5	784	-	-	-	-	1	2	3	5
A-n33-k5	661	-	-	-	-	1	1	1	1
A-n33-k6	742	-	-	-	0.13	26	26	7	-1
A-n45-k6	944	2.01	3.39	0.53	0.42	-1	-1	-1	-1
A-n46-k7	914	-	-	-	-	13	1	2	4
A-n54-k7	1167	0.43	0.60	-	-	-1	-1	2	31
A-n55-k9	1073	-	-	-	-	20	6	2	10
A-n65-k9	1174	0.34	0.68	0.34	0.85	-1	-1	-1	-1
A-n69-k9	1159	0.69	0.69	1.21	0.43	-1	-1	-1	-1
A-n80-k10	1763	1.13	1.36	0.85	0.91	-1	-1	-1	-1

Table 4: Results obtained for some class A benchmark instances comparing different methods to get an initial solution: (1) CP/LR scheme + VND, (2) CP/LR scheme without VND, (3) RCWS and (4) RCWS forced to use the minimum number of required vehicles.

Problem	BKS	Init.Sol.	CPU (s)	Fin. Sol.	CPU (s)	Gap BKS-FS (%)	# it
B-n44-k7	909	1000	0.01	909	98.85	-	3
B-n45-k5	751	768	0.06	751	56.59	-	2
B-n50-k7	741	778	0.00	741	47.18	-	1
E-n51-k5	521	597	0.01	521	903.22	-	28
E-n76-k8	735	792	0.04	736	1339.49	0.14	-1
E-n101-k8	817	890	0.04	829	2508.08	1.47	-1
F-n45-k4	724	777	0.00	724	264.81	-	2
F-n135-k7	1162	1299	0.07	1166	205415.08	0.34	-1
G-n262-k25	5685	5839	0.56	**5574**	175593.98	**-1.95**	1
M-n101-k10	820	857	0.03	820	2843.71	-	4
M-n151-k12	1015	1124	0.05	1025	18218.83	0.99	-1
P-n40-k5	458	513	0.00	458	22.53	-	1
P-n50-k7	554	581	0.00	554	67.50	-	2
P-n55-k8	588 (8)	606	0.01	**583 (7)**	101.71	**-0.85**	1
P-n65-k10	792	853	0.01	792	1420.63	-	24
P-n70-k10	827	881	0.01	830	1829.44	0.36	-

Table 5: Results obtained for some benchmark instances (improved results in bold)