A GRID-BASED HYBRID CELLULAR GENETIC ALGORITHM FOR VERY LARGE SCALE INSTANCES OF THE CVRP

Bernabé Dorronsoro
Dept. of Comp. Science
E.T.S.I. Informática
University of Málaga
29071 Málaga, SPAIN
Email: bernabe@lcc.uma.es

Daniel Arias
Dept. of Comp. Science
E.T.S.I. Informática
University of Málaga
29071 Málaga, SPAIN
Email: bernabe@lcc.uma.es

Francisco Luna
Dept. of Comp. Science
E.T.S.I. Informática
University of Málaga
29071 Málaga, SPAIN
Email: flv@lcc.uma.es

Antonio J. Nebro
Dept. of Comp. Science
E.T.S.I. Informática
University of Málaga
29071 Málaga, SPAIN
Email: antonio@lcc.uma.es

Enrique Alba
Dept. of Comp. Science
E.T.S.I. Informática
University of Málaga
29071 Málaga, SPAIN
Email: eat@lcc.uma.es

KEYWORDS
Cellular genetic algorithm, grid computing, VRP

ABSTRACT

This work presents a hybrid genetic algorithm (GA) for solving the largest existing benchmark instances of the capacitated vehicle routing problem (CVRP). The population of the algorithm is structured by following two classical parallelization models for GAs: coarse- and fine-grained. Indeed, the proposed model is a distributed GA (coarse-grained) in which each island is a cellular GA (fine-grained). It has been called PEGA (Parallel cEllular Genetic Algorithm). PEGA has been built on top of ProActive and it has been executed on a grid platform composed of more than 100 machines so as to reduce the computation time. The results show that, for many of the considered instances, PEGA improves the best results reported by any existing algorithm in the literature.

INTRODUCTION

The vehicle routing problem (VRP) (Dantzing and Ramster 1959) lies in minimizing the cost of a fleet of vehicles serving a set of customers from a unique depot (Fig. 1). Reducing this cost means minimizing the number of used vehicles and the length of their routes as well. This problem is of great interest due to two main facts. On one hand, the VRP is very interesting from an academic viewpoint because of its complexity (it is an NP-hard problem (Lenstra and Kan 1981)), the constraints it includes, and the many different existing versions. On the other hand, it has a direct application to the real world since it can be used by both large logistic enterprises and small local delivering companies. Indeed, using computer methods in such a business could often lead important cost savings to be reached. In many cases, these savings mean 20% of the total cost of the product (Toth and Vigo 2001).
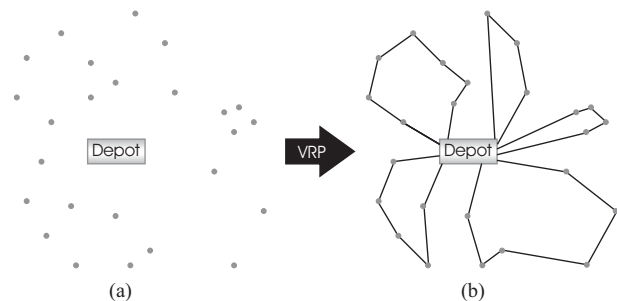

Fig. 1. The Vehicle Routing Problem consists in serving a set of geographically distributed customers (points) from a depot (a) using the minimum cost routes (b)

In the recent history of VRP, a constant evolution exists in the quality of the methodology used for solving the problem. These techniques comprise exact algorithms as well as heuristic methods. However, due to the complexity of the problem, there is no exact method capable of solving instances with more than 50 customers (Toth and Vigo 2001; Golden et al. 1998). It is also clear that generic heuristics cannot compete, in terms of solution quality, against current techniques such as those described in (Toth and Vigo 2001; Cordeau et al. 2005), which are specifically developed for solving VRP. Moreover, the potential search of some of these modern techniques, e.g. genetic algorithms (GAs), is not still fully exploited, especially when combining them with effective local search mechanisms. All these considerations could allow us to improve the search capability of a given algorithm.

This work is therefore focussed on GAs. Due to their population-based approach, GAs are very suitable for parallelization because their main operations (i.e., crossover, mutation, local search, and function evaluation) can be performed independently on different individuals. As a consequence, the performance of population-based algorithms is specially improved when run in parallel. Two parallelizing strategies are especially relevant for population-based algorithms: (1) parallelization of computation, in which the operations commonly
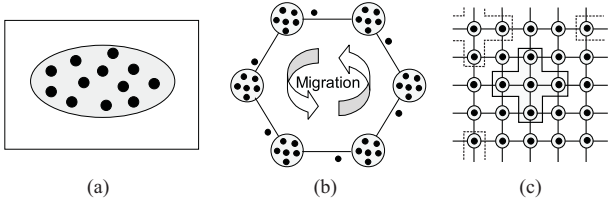
Fig. 2. Panmictic (a), distributed (b), and cellular (c) GAs

applied to each individual are performed in parallel, and (2) parallelization of population, in which the population is split in different parts, each one evolving in semi-isolation (individuals can be exchanged between subpopulations). Among the most widely known types of structured GAs, the *distributed* (dGA) (or coarse-grain) and *cellular* (cGA) (or fine-grain) algorithms are very popular optimization procedures (see Fig. 2). The parallelization of the population strategy is specially interesting since it does not only allow to speed up the computation, but also to improve the search capabilities of GAs (Alba and Tomassini 2002; Cantú-Paz 2000).

We propose in this work a new parallel cGA called PEGA (Parallel cEllular Genetic Algorithm). PEGA adopts all the parallelization strategies described above: the population is divided into several islands (dGA), the population of each island being structured by following the cellular model (cGA). Furthermore, each cGA implements a master/slave approach (parallelization of the computation) in order to compute the most costly operations applied to their individuals. Periodically, cGAs exchange information (migration) with the goal of inserting some diversity into their populations, thus avoiding them falling into local optima. By using such structure in the population, we keep a good balance between exploration and exploitation, thus improving the capability of the algorithm to solve complex problems (Alba and Tomassini 2002; Alba and Dorronsoro 2007). Additionally, cGAs in PEGA are hybridized with a local search method which is applied to every newly generated solution.

When solving very large problem instances, even heuristic methods like PEGA require a large amount of computational resources. This is exactly the context in which PEGA was developed because it is ultimately aimed at solving the largest instances of CVRP (a subclass of VRP): the VLSVRP benchmark (Li et al. 2005). With this goal in mind, PEGA has been enabled to run in *grid* computing platforms. This way, it has been implemented in ProActive (ProActive 2007), a Java GRID middleware library for an easy programming in *grid* environments. ProActive provides mechanisms for creating and managing collaborating objects (called *active objects*) which can be executed on remote machines. The remote access to an active object is carried out transparently by ProActive by using Java RMI.

The main contribution of this work is therefore the design of a new hybrid cGA which runs in parallel on *grid* computing platforms. This algorithm has been used to solve the largest known benchmark instances of the CVRP (Li et al. 2005) and it is able to improve the best known solutions computed by an optimization algorithm for most of the studied instances.

The paper is structured as follows. In the next section, we mathematically define the CVRP problem. Next, we describe PEGA, our proposal for solving large instances of the CVRP. Finally, the parameterization of the algorithm, the benchmark used, and the results obtained, as well as our conclusions and the main lines of future work are given in the two last sections.

THE VEHICLE ROUTING PROBLEM

The VRP can be defined as an integer programming problem which falls into the category of NP-hard problems (Lenstra and Kan 1981). Among the different variants of VRP we work here with the Capacitated VRP (CVRP), in which every vehicle has a uniform capacity of a single commodity. The CVRP is defined on an undirected graph $G = (\vec{V}, \vec{E})$ where $\vec{V} = \{v_0, v_1, \ldots, v_n\}$ is a vertex set and $\vec{E} = \{(v_i, v_j)/v_i, v_j \in \vec{V}, i < j\}$ is an edge set. Vertex v_0 stands for the *depot*, and it is from where m identical vehicles of capacity Q must serve all the *cities* or *customers*, represented by the set of n vertices $\{v_1, \ldots, v_n\}$. We define on E a non-negative *cost*, *distance* or *travel time* matrix $C = (c_{ij})$ between customers v_i and v_j. Each customer v_i has non-negative demand of goods q_i and drop time δ_i (time needed to unload all goods). Let be $\vec{V_1}, \ldots, \vec{V_m}$ a partition of $\vec{V}$, a route $\vec{R_i}$ is a permutation of the customers in $\vec{V_i}$ specifying the order of visiting them, starting and finishing at the depot v_0. The cost of a given route $\vec{R_i} = \{v_0, v_1, \ldots, v_{k+1}\}$, where $v_j \in \vec{V}$ and $v_0 = v_{k+1} = 0$ (0 denotes the depot), is given by:

$$\text{Cost}(\vec{R_i}) = \sum_{j=0}^{k} c_{j,j+1} + \sum_{j=0}^{k} \delta_j \ , \qquad (1)$$

and the cost of the problem solution ($\vec{S}$) is:

$$F_{\text{CVRP}}(\vec{S}) = \sum_{i=1}^{m} \text{Cost}(\vec{R_i}) \ . \qquad (2)$$

The CVRP lies in determining a set of m routes (i) of minimum total cost (see Equation 2); (ii) starting and ending at the depot v_0; and such that (iii) each customer is visited exactly once by exactly one vehicle; subject to the restrictions that (iv) the total demand of any route does not exceed Q ($\sum_{v_j \in \vec{R_i}} q_j \leq Q$); and (v) the total duration of any route is not larger than a preset bound D ($\text{Cost}(\vec{R_i}) \leq D$).

PEGA: PARALLEL CELLULAR GENETIC ALGORITHM

PEGA is a parallel GA in which the population is structured at two levels and in two different ways. As it can be seen in Fig. 3, the population is decentralized in a first
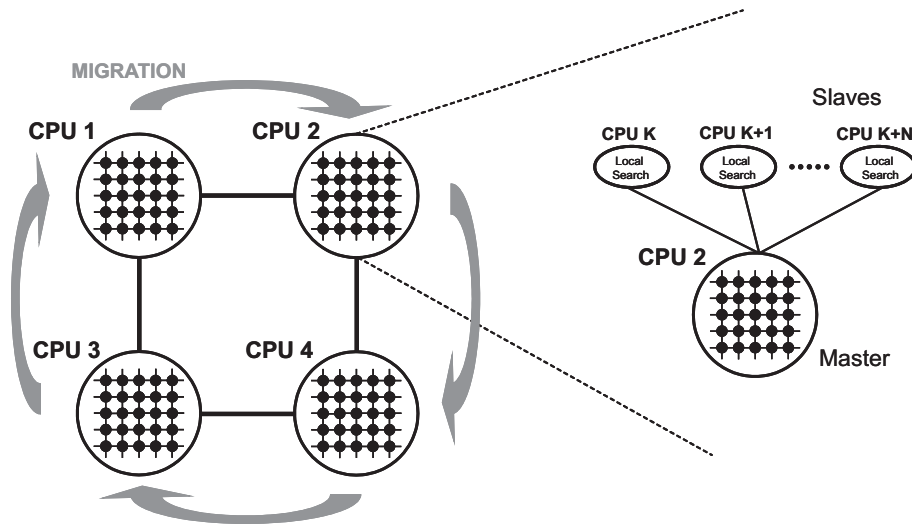
Fig. 3. Structure of PEGA

level by using the island model. The topology of these islands is an unidirectional ring so that migrations only occur between immediate neighbors. At the second level, each island is a cGA (see Section) in which the population follows a fine-grained parallelization strategy. Furthermore, each cGA is a master process that sends individuals to slave processes where they undergo the local search method, the task demanding the higher computational costs in the reproductive cycle of the cGA. This model and its parallel implementation in clusters were initially proposed in (Alba 1999).

Cellular Genetic Algorithm

This section describes the cGA used in each island of PEGA. Algorithm 1 shows its pseudocode. The population is structured in a 2D toroidal grid where a neighborhood structure is defined. The algorithm operates iteratively on each individual of the population (lines 3 and 4). The individuals can only interact with their nearby neighbors (line 5) and the parents are therefore chosen from the neighborhood of the current individual (line 6) with a given criterion. The recombination and mutation operators are applied in lines 7 and 8 with probabilities P_c and P_m, respectively. After that, the resulting individual undergoes a local search phase (line 9), and next its fitness value is computed (i.e., the cost of the represented

solution). The local search method is explained below. Finally, the best individual between the current and the offspring is placed in the equivalent position of an auxiliary population (line 11).

Once the reproductive cycle is applied to all the individuals of the population, the current population is replaced by the auxiliary one (line 14) and we then calculate some statistics (line 15). This loop is repeated until a termination condition is met.

Next, we detail some major issues concerning the implementation of the cGA used:

- **Individual representation.** We have used GVR (*Genetic Vehicle Representation*) (Pereira et al. 2002) for encoding the individuals. GVR uses a permutation of integer numbers which contains both customers and route splitters (delimiting different routes). Each route is composed of the customers between two route splitters in the individual. No unfeasible individuals are allowed so when either the maximum capacity of the vehicles or the maximum route length are exceeded, a repair procedure is executed so that it splits routes into two or more different subroutes which do not violate any constraint. This repair procedure is very fast and it also introduces a high level of diversity into the population. This represents an important point since in our previous studies (not using GVR) the population diversity was lost at the end of the evolution (Alba and Dorronsoro 2004; Alba and Dorronsoro 2007).
- **Generation of the initial population.** The individuals of the initial population are randomly generated, and then they are modified as shown in (Pereira et al. 2002) with the goal of obtaining feasible solutions.
- **Recombination operator.** PEGA uses the *generic crossover* originally proposed in (Pereira et al. 2002). The main feature of this operator is to promote diversity in the population. This is a very important feature since in our previous experiences addressing this problem (Alba and Dorronsoro 2007; Alba and Dorronsoro 2004; Alba

---

Algorithm 1 Pseudocode of a cGA

```
 1: proc Steps_Up(cga)        //Algorithm parameters in 'cga'
 2: while not Termination_Condition() do
 3:   for x ← 1 to WIDTH do
 4:     for y ← 1 to HEIGHT do
 5:       n_list←Get_Neighborhood(cga,position(x,y));
 6:       parents←Selection(n_list);
 7:       aux_indiv←Recombination(cga.Pc,parents);
 8:       aux_indiv←Mutation(cga.Pm,aux_indiv);
 9:       aux_indiv←Local_Search(cga.Pl,aux_indiv);
10:       Evaluate_Fitness(aux_indiv);
11:       Replacement(position(x,y),aux_indiv,cga,aux_pop);
12:     end for
13:   end for
14:   cga.pop←aux_pop;
15:   Update_Statistics(cga);
16: end while
17: end_proc Steps_Up;
```

Algorithm 2 Pseudocode of the recombination operator

// Let I_1 and I_2 be the chosen parents from the neighborhood;
Choose a random subroute $SR = \{a_1, \ldots, a_n\}$ de I_2
Find the customer $c \notin SR$ geographically closest to a_1
Remove all the customers from I_1 that are included in SR
The offspring is obtained after inserting SR into the genetic material of I_1 so that a_1 is placed just after c
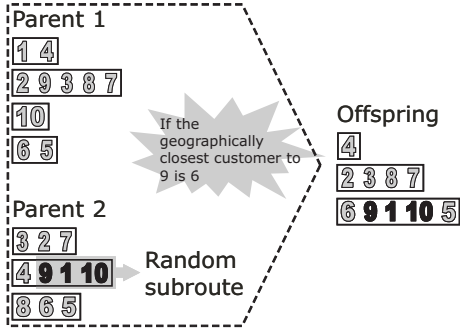

Fig. 4. Recombination operator used: *Generic crossover*

and Dorronsoro 2006), the entire population converges towards the same local optimum many times. This operator is somewhat unusual because the newly generated offspring does not only include genetic material from the two parents but also random components, as shown in Fig. 4. Algorithm 2 outlines the crossover operator used.

• **Mutation.** The mutation phase is composed of four different mutation operators which are applied with different probabilities (only one of them is applied each time), as proposed in (Pereira et al. 2002). Using these four mutation procedures allows us to modify the itinerary of a route, to move customers between routes, and to add or remove routes. They are (see Fig. 5):

– **Swap.** It swaps the position of two randomly chosen customers (belonging to the same route or not).

– **Inversion.** It reverses the visiting order of the customers between two randomly selected points of the permutation. In this case, all the customers must be in the same route.

– **Insertion.** It selects a gene (either customer or route splitter) and inserts it in another randomly selected place of the same individual.

– **Dispersion.** It is similar to the Insertion operator, but it is applied to a subroute (set of customers) rather than to a single customer.
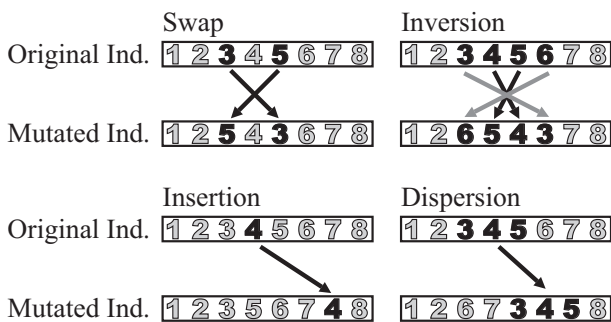

Fig. 5. The mutation operators

• **Local search.** The local search method applies up to 50 steps of *1-Interchange* (Osman 1993) and then up to 50 steps of *2-Opt* (Croes 1958) to each route of the solution reached by *1-Interchange*. These values were set after a tuning process. The *1-Interchange* method lies in interchanging a customer from a route by other customer belonging to other route, or inserting a customer from a route in a different route. On the other hand, *2-Opt* always works on one single route. It removes two edges of one route and connects the customers in the possible way (see Fig. 6). Since these two local search methods are deterministic, the search stops if no improvements have been reached in one single step. This will allow us to largely reduce the execution times.
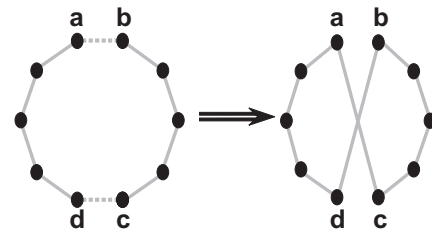

Fig. 6. The 2-Opt operator

EXPERIMENTATION

Recently, Li, Golden and Wasil presented in (Li et al. 2005) a new set of instances for the CVRP which are mainly characterized by the high number of customers used. This set of problems was called VLSVRP or *Very Large Scale VRP*. The size of the proposed instances in VLSVRP ranges between 560 and 1200 customers, whereas the most widely used and accepted benchmarks in the research community up to now were composed of instances between 50 and 199 customers in the CMT case (Christofides et al. 1979), or problems between 200 and 483 customers in (Golden et al. 1998). Some additional important characteristics of VLSVRP are: it has been generated by using an instance generator, thus easing the creation of instances of larger size (the reader is referred to (Li et al. 2005) for more details on the instance generator); the generated instances are geometrically symmetric with a circular pattern, which allows the best solution to be visually estimated; and the instances of the VLSVRP all include constraints on the maximum length of the routes.

PEGA has been designed with the aim of being used to solve the VLSVRP instances. Because of the complexity and large size of these problems, PEGA has been executed on a grid composed of up to 125 heterogeneous computers (PCs and Sun workstations/servers). The algorithm is implemented in Java, using ProActive to manage all the grid related issues. The parameterization of PEGA used for the experiments is detailed in Table I. Concretely, PEGA is composed of 4 islands arranged in a unidirectional ring. Each island operates the cGA described above in Section , using a square toroidal grid of
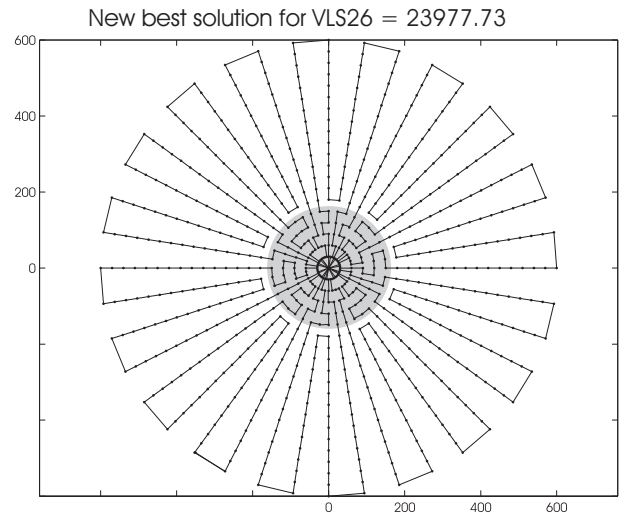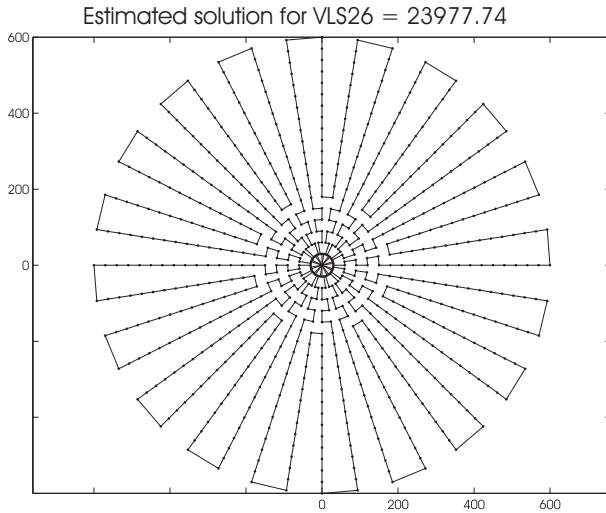
Fig. 7. New best solution for VLS26. Differences between the new and the previous solution can be noticed in the shadowed area

TABLE I: Parameterization used in PEGA

Population size	Islands: 100 Individuals (10×10)
	Total: 400 Individuals (4 islands)
Neighborhood	NEWS
Parent Selection	Binary tournament + Current Individual
Recombination	*generic crossover* (Pereira et al. 2002), $p_c = 1.0$
	Swap ($p_{int} = 0.05$),
Mutation	Inversion ($p_{inv} = 0.1$),
	Insertion ($p_{ins} = 0.05$),
	and Dispersion ($p_{disp} = 0.15$)
Replacement	Replace if better
Local Search	1-Interchange + 2-Opt,
	50 optimization steps each
Migration Frequency	Every 10^4 evaluations
Stopping Condition	500,000 evaluations in each island

10×10 individuals. In the reproductive cycle, one parent is chosen by binary tournament in the neighborhood of the current individual (this neighborhood is composed of the individuals in the North, East, West, and South – NEWS). The other parent is the current individual itself. The two parents are always recombined ($p_c = 1.0$) by using the *generic crossover* operator. The resulting individual undergoes mutation by one of the operators Swap, Inversion, Insertion, and Dispersion with probabilities 0.05, 0.1, 0.05, and 0.15, respectively (Pereira et al. 2002). Next, the local search method is applied to the mutated individual. As explained before, the method firstly executes 50 steps of *1-Interchange* (exploring combinations of customers among different routes) and then it applies 50 steps of *2-Opt* so as to optimize separately each newly generated route. The resulting offspring replaces the current individual in the population if the former has a better fitness value than the latter.

In the proposed implementation, migration takes place every 10^4 evaluations. At each migration operation, the islands send their best individual to their immediate neighbor. When a subpopulation receives a migrant, it replaces the worst individual in the local population.

Table II presents the results of PEGA for the VLSVRP benchmark (the best result for each instance is marked

in **boldface**). The table includes the name of the studied instances, their sizes (number of customers to be served), the best known solution for each instance (BNS) (Li et al. 2005), the best and average solutions found by PEGA, and the average execution time it takes. The fitness values of the solutions represent the cost in terms of the global distance traversed by the vehicles.
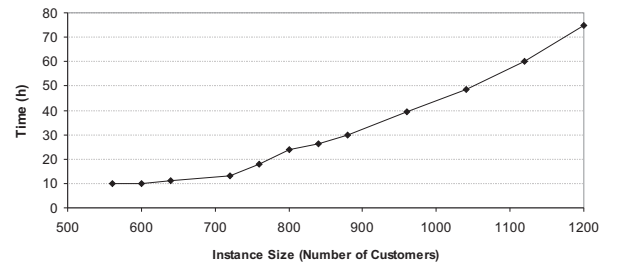

Fig. 8. Computation times of PEGA for different problem sizes

The values in Table II have been obtained after performing four independent runs of the smaller instances (VLS21 to VLS25) and two independent runs of the remaining instances. This small number of independent runs is due to the very long computational times required by PEGA to solve the problem, which ranges from 10 hours for the smaller instances to 72 hours in the case of VLS32. This is motivated by two main issues: (i) on one hand, the local search is a very high computational demanding task, which grows almost exponentially with the size of the solved instance; (ii) on the other hand, since the optimal solution is unknown, the stopping condition is to reach a preprogrammed number of function evaluations, and this number has to be large enough so that PEGA shall be able to find or even overcome the best known solution. Figure 8 shows an evolution of the computational times with the size of the instances. As it can be seen, the grown curve of these times over the increasing problem size is more than linear.

It is remarkable the accurate results despite the low

TABLE II: Results of PEGA for VLSVRP

Instance	Size	BNS	Best	Average	Time (h)
VLS21	560	**16212.83**§	**16212.83**	$16212.83_{\pm 4.42e-4}$	10.08
VLS22	600	**14641.64**†	14652.28	$14755.90_{\pm 98.88}$	10.08
VLS23	640	**18801.13**§	**18801.13**	$18801.13_{\pm 4.32e-6}$	11.28
VLS24	720	**21389.43**§	**21389.43**	$21389.43_{\pm 7.63e-6}$	13.20
VLS25	760	**17053.26**§	17340.41	$17423.42_{\pm 72.10}$	18.00
VLS26	800	23977.74§	**23977.73**	$23977.73_{\pm 3.49e-5}$	23.76
VLS27	840	**17651.60**†	18326.92	$18364.57_{\pm 37.66}$	26.40
VLS28	880	**26566.04**§	**26566.04**	$26566.04_{\pm 1.33e-6}$	30.00
VLS29	960	**29154.34**§	**29154.34**	$29154.34_{\pm 4.24e-5}$	39.60
VLS30	1040	**31742.64**§	31743.84	$31747.51_{\pm 3.67}$	48.72
VLS31	1120	**34330.94**§	**34330.94**	$34331.54_{\pm 0.60}$	60.00
VLS32	1200	**36919.24**§	37423.94	$37431.73_{\pm 7.79}$	74.88

§ Visually estimated solution (Li et al. 2005); † ORTH (Li et al. 2005)

Best Solutions Found
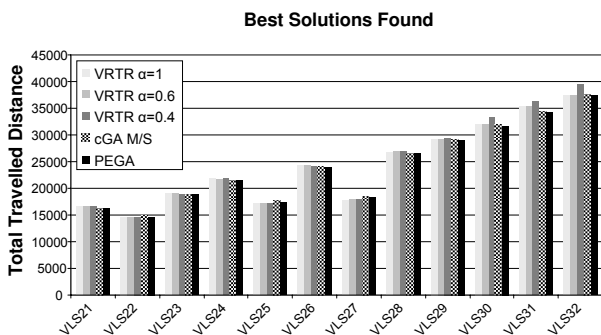
Fig. 9. PEGA vs. the state-of-the-art for solving the VLSVRP instances

TABLE III: Difference (%) between the best known solution and the results of PEGA and the algorithms of the state-of-the-art

Instance	VRTR (α value)			cGA M/S	PEGA
	(1.0)	(0.6)	(0.4)		
VLS21	2.41	2.56	3.25	0.02	**0.00**
VLS22	**0.07**	0.10	0.19	2.17	**0.07**
VLS23	1.09	1.56	0.20	**0.00**	**0.00**
VLS24	1.85	1.06	2.54	$5.61e-3$	**0.00**
VLS25	0.58	0.65	**0.55**	3.53	1.68
VLS26	0.88	0.93	0.13	0.05	**0.00**
VLS27	**0.97**	1.61	1.42	4.83	3.83
VLS28	0.15	0.82	0.83	**0.00**	**0.00**
VLS29	0.09	0.10	0.85	0.02	**0.00**
VLS30	0.74	0.69	4.74	0.64	**3.78e − 3**
VLS31	3.02	2.98	5.85	0.35	**0.00**
VLS32	1.36	**1.33**	6.76	2.02	1.37
Average	1.10	1.20	2.28	1.14	**0.58**

number of independent runs which have been carried out. Indeed, PEGA reaches the best known solution in 7 out of the 12 studied instances. Furthermore, PEGA has been able to improve the best known solution for VLS26. Even though the difference is very small, they represent very different solutions, as it can be seen in the shadowed area of Fig. 7. We also want to note that the best known solutions for all the instances have been visually estimated by using the geometric properties of the problems, and no algorithm was able to find them up to this work. In the case of the instances VLS22 and VLS27, the solutions were found by different RTR (*Record-To-Record*) algorithms that were tried in the development phase of VRTR (an enhanced version of RTR) (Li et al. 2005). However, neither details on these algorithms nor references to related works are given in (Li et al. 2005) for further reading on this topic. In Table II, ORTR means Other RTR algorithms.

Figure 9 shows a comparison between the best solutions obtained by PEGA, a master/slave cGA (called cGA M/S) with the same configuration used in the cGAs of the islands of PEGA (except for the termination condition, which is set to 200,000 evaluations in this case), and the three algorithms which are the state-of-the-art for VLSVRP. These three algorithms are different parameterizations of VRTR proposed in (Li et al. 2005). As

it can be seen, PEGA gets equal or lower fitness values (better results) than cGA M/S for all the tested instances. Regarding the three VRTR versions, PEGA outperforms them in all the instances but VLS25, VLS27, and VLS32. In Table III, we present the comparison among the three VRTR algorithms taken from the literature (Li et al. 2005), PEGA, and cGA M/S. The comparison is made for all the problem instances in terms of the difference (percentage) between the solution they reported and the best known solution (best values are in **boldface**). This value can be understood as a quality measure of the results obtained by the algorithms. From the results, it is noticeable that PEGA is a more robust algorithm with respect to the other compared approaches in the studied benchmark since it gets the best results in 9 out of the 12 VLSVRP instances. Additionally, PEGA obtains the best known solution in 7 instances, solutions which have never been reached by any algorithm, as stated before. If we compare PEGA with the cGA M/S we can notice that structuring the population in two different levels (fine- and coarse-grained) is highly beneficial versus using just one level of decentralization (the fine-grained model of cGA M/S) for the tested problems. Indeed, PEGA outperforms our cGA M/S for all the tested instances.

The last row in Table III shows the average of the differences between the solutions found by each algorithm and the best known solutions of the 12 VLSVRP instances. As it can be seen, PEGA gets the lowest (best) value among the four compared algorithms. Indeed, this value is half the value of the best VRTR approach ($\alpha = 1.0$).

CONCLUSIONS AND FUTURE WORKS

This work presents a very powerful algorithm for solving extremely hard instances of CVRP. The proposed algorithm, called PEGA, is a distributed GA with four islands, in which each island is in turn a cellular GA. The population is therefore structured at two levels. Additionally, and because of the complexity of the studied instances, PEGA has been parallelized using ProActive so that it can be executed on grid computing platforms. The parallelization strategy used in each island follows a master/slave scheme as well.

PEGA has been compared against the algorithms of the state-of-the-art for the benchmark of the CVRP which comprises the largest instances, the set of problems VLSVRP. As a result, PEGA not only was able to find the best results in 9 out of the 12 instances, but also it computed the best known solution for 7 instances (never found before by any algorithm, but they have been visually estimated by using their geometric features). PEGA also found a new best solution for the VLS26 instance.

As very near future work, we are working on increasing the number of independent runs of our experiments, because we want to provide the results with statistical confidence. We are also planing to reduce the computational times by increasing the size of our grid computing platform as well as by improving the algorithm to make it more efficient. In this second issue, our idea is to focus on developing new improved local search methods, because it is the most computationally costly step of the reproductive cycle.

ACKNOWLEDGEMENTS

REFERENCES

Alba, E. 1999. "Análisis y diseño de algoritmos genéticos paralelos distribuidos," Ph.D. dissertation, Universidad de Málaga, Málaga.

Alba, E. and Dorronsoro, B. 2004. "Solving the vehicle routing problem by using cellular genetic algorithms," in *Evolutionary Computation in Combinatorial Optimization – EvoCOP04*, ser. LNCS, J. Gottlieb and G. R. Raidl, Eds., vol. 3004. Coimbra, Portugal: Springer Verlag, 11–20.

Alba, E. and Dorronsoro, B. 2006. "Computing nine new best-so-far solutions for capacitated VRP with a cellular genetic algorithm," *Information Processing Letters*, 98(6):225–230.

Alba, E. and Dorronsoro, B. 2007. *Engineering Evolutionary Intelligent Systems*, ser. Studies in Computational Intelligence. Springer-Verlag, ch. 13, "A Hybrid Cellular Genetic Algorithm for the Capacitated Vehicle Routing Problem".

Alba, E. and Tomassini, M. 2002. "Parallelism and evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, 6(5):443–462.

Cantú-Paz, E. 2000. *Efficient and Accurate Parallel Genetic Algorithms*, 2nd ed., ser. Book Series on Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, vol. 1.

Cordeau, J.; Gendreau, M.; Hertz, A.; Laporte, G. and Sormany J. 2005 *New Heuristics for the Vehicle Routing Problem*. Logistics Systems: Design and Optimization, A. Langevin and D. Riopel, eds., Springer, New York, 279–297.

Christofides, N.; Mingozzi, A. and Toth, P. 1979. *Combinatorial Optimization*. John Wiley, 1979, ch. "The Vehicle Routing Problem," 315–338.

Croes, G. 1958. "A method for solving traveling salesman problems," *Operations Research*, 6:791–812.

Dantzing, G. and Ramster, R. 1959. The truck dispatching problem. *Management Science* 6, 80–91.

Golden, B.; Wasil, E.; Kelly, J. and Chao I.-M. 1998. *Fleet Management and Logistics*. Boston: Kluwer, 1998, ch. "The Impact of Metaheuristics on Solving the Vehicle Routing Problem: algorithms, problem sets, and computational results," 33–56.

Lenstra, J. and Kan, A.R. 1981. "Complexity of vehicle routing and scheduling problems," *Networks*, vol. 11, 221–227.

Li, F.; Golden, B. and Wasil, E. 2005. "Very large-scale vehicle routing: New test problems, algorithms, and results," *Computers & Operations Research*, vol. 32, 1165–1179.

Osman, I. 1993. "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problems," *Annals of Operations Research*, 41:421–451.

Pereira, F.; Tavares, J.; Machado, P. and Costa, E. 2002. "GVR: a new representation for the vehicle routing problem," in *13th Irish Conference Proceedings on Artificial Intelligence and Cognitive Science (AICS)*, M. O. et al., Ed. Ireland: Springer-Verlag, 95–102.

"ProActive official web site," http://www-sop.inria.fr/oasis/proactive/

Toth, P. and Vigo, D. 2001. *The Vehicle Routing Problem*, ser. Monographs on Discrete Mathematics and Applications, P. Toth and D. Vigo, Eds. Philadelphia: SIAM.