applied sciences

MDPI

Article

# A Grid-Based Genetic Approach to Solving the Vehicle Routing Problem with Time Windows

**Marco Antonio Cruz-Chávez [1,*], Abelardo Rodríguez-León [2], Rafael Rivera-López [2] and Martín H. Cruz-Rosales [3]**

1 Research Center in Engineering and Applied Sciences, Autonomous University of Morelos State (UAEM), Avenida Universidad 1001 Colonia Chamilpa, C.P. 62209 Cuernavaca, Morelos, Mexico
2 Department of Systems and Computing, México National Technological/I.T. Veracruz, Calzada Miguel Ángel de Quevedo 2779, C.P. 91860 Veracruz, Mexico
3 Faculty of Accounting, Administration & Informatics, UAEM, Avenida Universidad 1001 Colonia Chamilpa, C.P. 62209 Cuernavaca, Morelos, Mexico
* Correspondence: mcruz@uaem.mx

Received: 24 July 2019; Accepted: 30 August 2019; Published: 4 September 2019


**Featured Application: This research allows working collaboratively between different institutions through a grid computing infrastructure, sharing supercomputing equipment to be able to find good solutions in a more efficient way to problems intractable that are of real application as the Vehicle Routing Problem.**

**Abstract:** This paper describes one grid-based genetic algorithm approach to solve the vehicle routing problem with time windows in one experimental cluster MiniGrid. Clusters used in this approach are located in two Mexican cities (Cuernavaca and Jiutepec, Morelos) securely communicating with each other since they are configured as one virtual private network, and its use as a single set of processors instead of isolated groups allows one to increase the computing power to solve complex tasks. The genetic algorithm splits the population of candidate solutions in several segments, which are simultaneously mutated in each process generated by the MiniGrid. These mutated segments are used to build a new population combining the results produced by each process. In this paper, the MiniGrid configuration scheme is described, and both the communication latency and the speedup behavior are discussed. Experimental results show one information exchange reduction through the MiniGrid clusters as well as an improved behavior of the evolutionary algorithm. A statistical analysis of these results suggests that our approach is better as a combinatorial optimization procedure as compared with other methods.

**Keywords:** networks; genetic algorithms; virtual private network; super-linear speedup; latency

## 1. Introduction

The vehicle routing problem (VRP) involves the definition of an optimal route set for several delivery vehicles satisfying the requirements of the customers. In particular, the VRP with time windows (VRPTW) is a VRP variant in which a slot of time is assigned for the delivery to the customer. This problem is known as one NP-hard combinatorial optimization problem, and many algorithms have been proposed in the existing literature to try to solve it, highlighting the use of swarm and evolutionary algorithms such as genetic algorithms and particle swarm optimization methods. In a genetic algorithm (GA), each candidate solution to VRPTW is encoded in one chromosome, and a population of them evolves by applying the selection, crossover, and mutation operators. This evolutionary process is guided by a fitness function, helping to reach a near-optimal solution. Since the

permutation characteristics of a VRPTW candidate solution, new chromosomes must represent only feasible solutions, but traditional recombination operators can create infeasible solutions with repeated values. In a previous work, Díaz-Parra and Cruz-Chávez [1] describe an approach named GA-VRPTW (Genetic Algorithm to Vehicle Routing Problem with Time Windows) evolving only feasible VRPTW routes by the use of modified genetic operators for selection, crossover, and mutation. Experimental results with several Solomon benchmark problems with 100 customers indicate that GA-VRPTW is very competitive compared with other GA-based algorithms and in some cases is even better. Otherwise, GA-VRPTW is a sequential algorithm whose performance analysis shows that the mutation operator requires the most computational time.

In this paper, one grid-based genetic algorithm (GGA) to improve the GA-VRPTW performance using an experimental MiniGrid is introduced. GGA uses the computing resources of two clusters that are geographically separated but linked together since they are part of a MiniGrid configured as one virtual private network (VPN). Based on the genetic operators proposed in the GA-VRPTW algorithm, GGA splits the population of chromosomes in several segments, which are simultaneously mutated in each process generated by the MiniGrid. After the application of the mutation operator in each MiniGrid node, the mutated segments are used to build a new population of improved chromosomes.

Since VRPTW is considered an NP-hard problem, many approaches have been proposed to find appropriate solutions, and its study has importance since several real-problems in manufacturing, scheduling, and logistics can be formulated as one vehicle routing problem. These approaches can be grouped into exact methods such as dynamic programming [2], branch-and-bound [3], and branch-and-cut [4], as well as metaheuristic-based algorithms such as Tabu Search (TS) [5], ant colony optimization (ACO) [6], simulated annealing (SA) [7], and evolutionary algorithms (EAs) [8–11].

Kohl et al. [12] introduce a set-partitioning-based approach known in the existing literature as the KDMSS algorithm, in which a Dantzig–Wolfe decomposition scheme is used in conjunction with a two-path cut procedure to reach better lower bounds to several VRPTW instances of the Solomon benchmark problems with 100 and 150 customers. Arbelaitz et al. [13] present a two-phase parallel SA algorithm to solve the VRPTW using a message-passing communication scheme. Experimental results of 56 Solomon benchmarks problems with 100 customers reach near-optimal solutions. Also, it was observed that increasing the number of processors improved the algorithm efficiency. A similar scheme was proposed by Wieczorek [14] where experimental results show that depending on the problem characteristics, some improve their efficacy as the number of processors increased, and with others, the quality decreased or remained the same regardless of the number of processors. Gehring and Homberger [15] introduce two parallel two-phase hybrid metaheuristics using TS and SA, designated as the HM4 and HM4C methods, improving the solution quality of several problems generated by the authors. Berger and Barkaoui [16] propose one parallel master-slave co-evolutionary approach using two populations with different fitness functions: The first minimizing the total travel distance and the second reducing the time window constraints violations. Le Bouthillier and Crainic [17] implement a combination of TS and GA running in several cooperative threads and using the population as shared memory. One GA in each thread is used to evolve the population using different selection and crossover operators, and TS is a local search method applied to improve the offspring quality. Ropke and Pisinger [18] apply the adaptive large neighborhood search (ALNS) heuristic to solve five different VRP variants. First, each variant is transformed in a rich pickup and delivery problem with time windows (RPDPTW), and then the ALNS is used to solve it. This algorithm was able to improve the best-known solution in 183 of 486 benchmark problems.

Furthermore, Nalepa and Czech [19] propose several thread-cooperation schemes to solve VRPTW instances: Two use a constant cooperation frequency rate, and the others use two adaptive cooperation rates. Experimental results with several Gehring and Homberger benchmark problems show that, for harder instances, the cooperation frequency increases, and higher solution accuracy is obtained, and this behavior is contrary for less hard instances. Nalepa and Blocho [20] propose an island-model-based approach for a parallel memetic algorithm (PMA-VRPTW) applying refinements to a population of

candidate solutions evolving in a parallel scheme. Experimental tests with a symmetric-multiprocessor cluster report better solutions than the best-known solutions of 19 Gehring and Homberger benchmark problems with 1000 customers. Baños et al. [21] present another island-based parallel method to solve the capacitated VRPTW named multiple temperature Pareto simulated annealing (MT-PSA). Experimental results with several Solomon benchmark problems show that the parallel version produces better solutions than those obtained by the sequential version of the algorithm. Barbucha [22] proposes one cooperative population learning algorithm (CPLA) using several search heuristics implemented as software agents cooperating in a two-stage procedure that first learn and then promote the best solutions. CPLA is evaluated using 56 Solomon benchmark problems with 100 customers, and its results are comparable to those produced by other methods. Jawarneh and Abdullah [23] apply the bee colony optimization (BCO) with one sequential insertion heuristic maintaining an acceptable population diversification rate to solve several Solomon benchmark problems. They apply a self-adapting tuning strategy to decide the search behavior of each bee in the swarm. Pierre and Zakaria [24] implement a multi-objective GA with a stochastic partially optimized cyclic shift crossover operator to build feasible candidate solutions in its evolutionary process guiding using the distance route and the number of vehicles as fitness functions. The performance of this method is evaluated using 56 Solomon benchmarks problems, and the results indicate that this approach produces 16 solutions comparable with the best-known solutions. Bychkov and Batsyn [25] propose a hybrid algorithm combining ACO and TS to solve the capacitated VRPTW. In this approach, a set of ants is used to find promising candidate solutions, and TS is applied to improve the quality of these solutions in each step of its iterative process.

Wang et al. [26] implement a message-passing communication scheme with a parallel SA (p-SA) algorithm, including one insertion-based heuristic to solve a VRP variant with simultaneous pickup and delivery with time windows. Experimental tests were performed with several Gehring and Homberger benchmark problems and with 65 Wang and Chen benchmark problems. Results show the algorithm effectiveness since it obtains better solutions for 12 Wang and Chen instances, and also reach the best-known solution for the remaining instances.

On the other hand, the use of grid environments to solve optimization problems has been scarcely documented in the existing literature. For example, Fujisawa et al. [27] describe the use of a clusters grid to solve several optimization problems such as the nonconvex quadratic optimization problem and the semidefinite problem, as well as to solve polynomial systems of equations, and Zunino et al. [28] introduce a general framework to implement exact optimization algorithms on a grid environment. In particular, Rodriguez-León et al. [29] implement a parallel GA using a two-stage communication scheme to share the population of candidate solutions in the nodes of an experimental grid with two clusters. First, the populations evolve in each cluster in independent form using point-to-point communication with send/receive methods provided by the MPI library, and then the information exchange between the clusters of the grid is achieved through the FTP protocol using only one node in each cluster. In general, algorithms implemented in grid environments can be classified into two groups: The first one include those developed over the middleware supporting a grid environment such as NetSolve [30], Condor [31,32], Globus [33], and gLite [34], and the other are those applied in grids configured as one VPN [29,35,36]. Each one has its advantages and disadvantages. In this work, GGA is implemented in an experimental MiniGrid configured as one VPN to solve VRPTW instances. The main contributions of this work are: (a) The proposed structure for the GGA including the migration scheme of mutated segments between grid nodes, (b) the communication scheme among the clusters of the MiniGrid, using collective communication methods provided by the MPI library, and (c) the use of the grid computing to efficiently solve complex problems. It can be used to solve problems using the total computing capacity (number of cores) of one virtual supercomputer (grid). The resolution of the NP-hard problems is intractable [37] due to the significant size problem, and the use of many computer resources is required to allow an extensive exploration of the solution space and one full exploitation of promising areas in relatively shorter times. Parallelization of algorithmic

processes can lead to improved algorithmic designs for the treatment of these problems. This impact is stronger where computing infrastructure is a constraint in the study of these problems, and the use of several computational clusters to build a grid environment is a viable option in the study of these complex problems.

The rest of this document is structured as follows: Section 2 describes the VRPTW and the application of metaheuristics to solve this problem. A description of the MiniGrid components is included in Section 3, and Section 4 describes the proposed approach for the GA parallelization and its implementation in a grid environment. Section 5 presents the experimental study. Finally, the results and discussion of this approach are summarized in Section 6.

## 2. Vehicle Routing Problem with Time Windows

VRP involves the definition of an optimal route set for several delivery vehicles sharing a central depot and satisfying the requirements of several geographically dispersed customers while minimizing the total routes distance or some other criterion [38]. This problem was first studied by Dantzig and Ramser [39] and is considered one NP-hard problem [37].

Several VRP variants have been proposed in the existing literature such as the VRPTW, the capacitated VRP (CVRP), and the multiple depots VRP (MDVRP), among others. In the VRPTW, every customer has to be supplied within a specified time window. In the CVRP, every vehicle has a limited capacity, and in the MDVRP, the vendor uses many depots to supply the customers.

### 2.1. Mathematical Formulation of VRPTW

Based on the formal definition introduced by Toth and Vigo [30], the VRPTW is presented in Equations (1)–(11). In this formulation, the problem is represented through a directed graph composed of a set of nodes containing the customers and the central depot. These nodes are joined by arcs symbolizing the possible route of a vehicle between two nodes. The graph is formally defined as $G(V, E)$, where $V = \{0, \ldots, n\}$ is the set of $n$ nodes, and $E$ is the set of edges joining pairs of nodes in $V$. Furthermore, $K$ is the set of delivery vehicles, and $N = V \backslash \{0\}$ is the set of customers, where the node 0 in $V$ is the vehicle depot.

On the other hand, $c_{ij}$ is the travel cost from the $i$-th node to the $j$-th node, and $x_{ijk}$ is equal to 1 if the arc joining the $i$-th node and the $j$-th is used by $k$-th vehicle, otherwise it is 0. $d_i$ is the demand of $i$-th customer, and $C$ is the vehicle capacity. Furthermore, $w_{ik}$ specifies the start service time of $i$-th customer by the $k$-th vehicle, $s_i$ is the service time of $i$-th customer, $t_{ij}$ is the time travel from the $i$-th node to the $j$-th node, and $a_i$ and $b_i$ are the time window limits of $i$-th node. $E$ and $L$ represent the earliest possible departure time and the latest possible arrival time at the depot, respectively. $\Delta^+(i)$ denotes the set of nodes directly reachable from the $i$-th node, and $\Delta(i)$ denotes the set of nodes from which the $i$-th node is reachable.

$$min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ijk} \tag{1}$$

subject to:

$$\sum_{k \in K} \sum_{j \in \Delta^+(i)} x_{ijk} = 1 \quad \forall i \in N \tag{2}$$

$$\sum_{j \in \Delta^+(0)} x_{0jk} = 1 \quad \forall k \in K \tag{3}$$

$$\sum_{i \in \Delta^-(j)} x_{ijk} - \sum_{i \in \Delta^+(j)} x_{jik} = 0 \quad \forall k \in K, j \in N \tag{4}$$

$$\sum_{i \in \Delta^-(n+1)} x_{i,n+1,k} = 1 \quad \forall k \in K \tag{5}$$

$$x_{ijk}\left(w_{ik} + s_i + t_{ij} - w_{jk}\right) \leq 0 \quad \forall k \in K, (i,j) \in A \tag{6}$$

$$a_i \sum_{j \in \Delta^+(i)} x_{ijk} \leq w_{ik} \leq b_i \sum_{j \in \Delta^+(i)} x_{ijk} \quad \forall k \in K, i \in N \tag{7}$$

$$E \leq w_{ik} \leq L \quad \forall k \in K, i \in \{0,n+1\} \tag{8}$$

$$\sum_{i \in N} d_i \sum_{j \in \Delta^+(i)} x_{ijk} \leq C \quad \forall k \in K \tag{9}$$

$$x_{ijk} \geq 0 \quad \forall k \in K, (i,j) \in A \tag{10}$$

$$x_{ijk} \in \{0,1\} \quad \forall k \in K, (i,j) \in A. \tag{11}$$

The objective function in Equation (1) is the minimization of the total travel cost. Furthermore, Equation (2) indicates that a single vehicle must serve only one customer. The path of each vehicle is represented by Equations (3)–(5) while Equations (6), (8), and (9) guarantee both the time feasibility and the capacity of the vehicles. Equation (7) indicates that one service should start in the time window [a, b]. Finally, Equation (10) are non-negativity constraints and Equation (11) indicate that variables must have binary values.

### *2.2. A Sequential Genetic Algorithm for Solving the VRPTW*

GAs are search procedures inspired by evolutionary theories synthesizing the Darwinian evolution through natural selection with the Mendelian genetic inheritance [40]. This type of evolutionary algorithm stands out for its robustness and its ability to carry out one intelligent search in large, complicated, and unpredictable search spaces. These algorithms can be applied to solve diverse problems by encoding the candidate solutions as sequences of genes, as well as by defining an appropriate fitness function. In particular, GA can be applied to solve the VRP by encoding the candidate solutions using a sequence of integer-based values, where each value represents one customer or one delivery vehicle [41].

Several studies have been carried out implementing one GA to solve VRP instances, where various encoding schemes and several genetic operators have been proposed. In particular, the GA-VRPTW method [1] is a sequential algorithm using the *best-selection* operator and two new recombination operators: *crossover-k* and mutation-s. These genetic operators are described in Table 1. Each gene in a chromosome represents one customer that must be visited by one vehicle. This gene has several associated values: The Euclidean distance between two customers, as well as the demand and the time window associated with one customer ($c_{ij}$, $d_i$, and $[a_i, b_i]$ in the previously mathematical formulation, respectively).

**Table 1.** Genetic operators in GA-VRPTW [1].

| Operator | Description |
|---|---|
| best-selection | This is a commonly used operator in literature. It selects the best chromosome from a population. |
| crossover-k | This operator generates two new offspring by making a random crossover of two chromosomes. *Crossover-k* operator randomly takes two numbers and carries out the crossover exclusively in the customer that corresponds to both chromosomes. This is done in order to avoid repetition and violation of time and capacity restrictions. |
| mutation-s | This operator is termed intelligent because it does not randomly make changes, rather it attempts to reduce the total travelled distance by making only changes that satisfy time and capacity constraints. This operator searches for a gene with a greater distance with respect to the previous customer, this is the *gene-candidate*. It searches for another gene with a shorter distance, this is the *gene-mutation*. Once the mutation-s operator has identified these genes, the *gene-mutation* makes the change if time and capacity constraints are not violated. The fitness is reduced due to the movements of genes in an offspring. |

The general scheme of the GA-VRPTW implementation is depicted in Figure 1. First, the initial population with only feasible solutions is created using the k-means clustering algorithm [42]. Next, the evolutionary process applies genetic operators to generate only feasible new solutions. Finally, the best candidate solution in the final population is selected as the result of the GA-VRPTW algorithm.

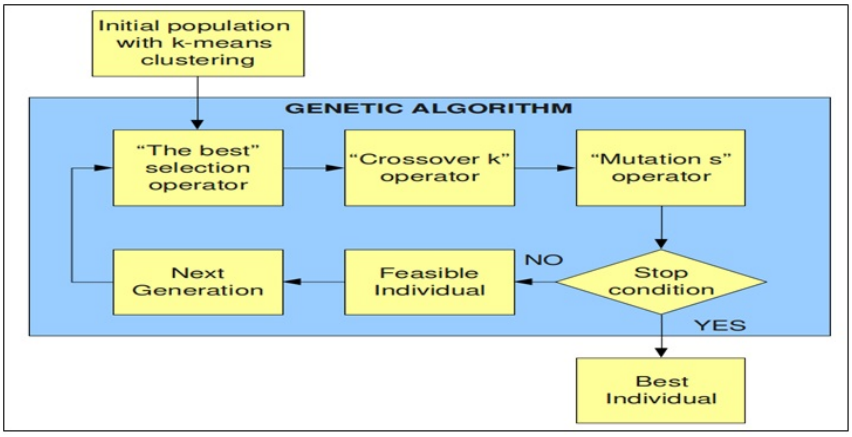


**Figure 1.** General scheme of the genetic algorithm vehicle routing problem with time windows (GA-VRPTW) algorithm [1].

In this approach, the mutation-s operator implements an iterated local search to reduce the total travel distance. This operator alters the order of the chromosome genes and ensures that this perturbation improves the fitness value of the mutated chromosome, and accomplishes this with the defined constraints (demand, time window, and distance). The use of the mutation-s operator in one chromosome with 10 genes is shown in Figure 2. In this example, the candidate genes to be mutated have a distance of 11.1 and 5.6, respectively. These genes are interchanged, the associated values are updated, and the new total travel distance is computed. If this is not a valid perturbation, other candidate genes must be selected. This local search procedure causes the computation time of the mutation operator to increase considerably as compared to the time used by the other genetic operators.

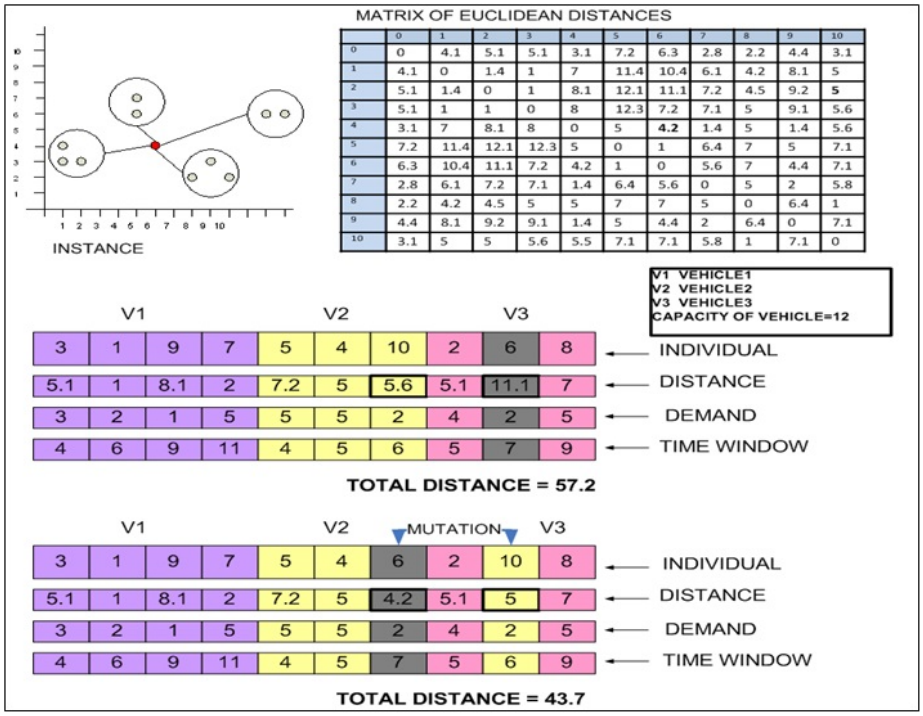


**Figure 2.** Mutation-s operator of the GA-VRPTW algorithm [1].

### 2.3. The Parallelization Approach as a Solution

The search to one optimum solution for the VRPTW is unattainable when the problem size grows since the execution time increases exponentially, given the NP-completeness characteristics of the problem [43]. Although metaheuristics are efficient to find reasonable solutions to complex problems, the single computer resources are not sufficient to reach results for these type of problems in a reasonable time. Parallelization is one of the most effective approaches to improve the performance of one algorithm and to overcome the single machine constraints.

Several parallel-based approaches to solve the VRPTW have been proposed, such as one parallel version of the Branch and Bound algorithm [44], and several metaheuristic-based approaches such as SA [14], TS [45], and PSO [46]. Furthermore, various parallel GAs have been described in the existing literature. Cantú-Paz [47] groups these approaches in four categories: Global master-slave, island, cellular, and hierarchical parallel GA (Table 2).

**Table 2.** Categories of parallel genetic algorithms [47].

| Category | Description |
|---|---|
| Global master-slave | In this approach, a single population is distributed among several nodes. Genetic operations are applied to the whole population. |
| Island GA | Several subpopulations evolve separately with the occasional migration of individuals between subpopulations. |
| Cellular GA | This approach consists of one spatially structured population. Selection and crossover are restricted to a small neighborhood. The neighborhoods are allowed to overlap, permitting some interaction among individuals. |
| Hierarchical parallel GA | This category combines an island model with either a master-slave or cellular GA. |

## 3. MiniGrid Infrastructures

Grid computing is an emerging technology where a set of heterogeneously networked resources distributed in geographically dispersed locations are coordinated to provide transparent, dependable, pervasive, and consistent computing support to diverse types of applications [48]. There are different levels of integration that range from the use of computing resources of two or more organizations to the extensive use of all grid resources [49].

There are several approaches to build and configure grid environments, and there are many efforts to define general schemes for grid resources exploitation. First, grid environments using Globus, Condor, or gLite are considered high-throughput grids or service grids. These grids manage the resource set (machines or clusters) and distribute the processes among these resources. The service grid's goal is to provide an abstraction level for the users so that they can use all the grid's resources while maintaining an adequate security level. Service grids can implement either sequential or parallel applications. On the other hand, the use of a VPN to cluster integration in grid environments avoids the need for specialized packages such as Globus or gLite [50]. A VPN-based grid can be considered a high-performance grid since it can run parallel applications using the resources of several clusters connected in the grid. A VPN creates an encrypted communication channel between groups to ensure secure data exchange.

In this work, the MiniGrid is composed of two clusters with homogeneous machines (Table 3), each belonging to a different organization, geographically distant (14.37 km), as is shown in Figure 3. One cluster is in the Autonomous University of Morelos State (UAEM), in Cuernavaca city, and the other in the Polytechnic University of Morelos State (UPEMOR), in Jiutepec city, both in the Mexican state of Morelos. In this case, OpenVPN is used to integrate the clusters in a grid environment. Communication between clusters is via a wireless WAN link between institutions (UAEM-UPEMOR) implemented via a point-to-point microwave link with ISM frequency bands with a bandwidth of 30 Mbs [51]. Message passing interface (MPI) is used to exchange data, and to reduce the communication rate between MiniGrid clusters. MPI has been used to develop applications with both Globus-based grids [52] and VPN-based environments [29,53,54].

**Table 3.** Infrastructure of hardware and software for the MiniGrid.

| **Software MiniGrid** | |
|---|---|
| **Centos Linux 5.5, 64 bits, Compiler gcc 4.1.2, OpenMPI1.8 MPICH2, Intel compiler MPI 12.0, Ganglia 3.1.7, NFS-utils 1.0.9, OpenVPN, Torque+Maui** | |
| Hardware for the UAEM cluster (cluster.cuexcomate.edu.mx) | Hardware for the UPEMOR cluster (cluster.texcal.edu.mx) |
| Switch 3COM 24/10/100/1000<br>Switch InfiniBand Mellanox, 18 ports, 40 Gb/s QDR | Switch 3COM 24/10/100/1000<br>Switch InfiniBand Mellanox, 18 ports, 40 Gb/s QDR |
| Master Node Motherboard: Two Intel Xeon, Six Core 3.06 GHz (12 cores, Total), 12MB cache, 6 HD 7200 RPM, 12 TB, 24 GB RAM, InfiniBand card 40 Gb/s | Master Node Motherboard: Two Intel Xeon, Six Core 3.06 GHz (12 cores, Total), 12 MB cache, 6 HD 7200 RPM, 12 TB, 24 GB RAM, InfiniBand card 40 Gb/s |
| 4 slave nodes, Node Motherboard: Two Intel Xeon Six core 3.06 GHz, 12 MB cache, 1 HD 7200 RPM, 500 GB, 24 GB RAM, InfiniBand card 40 Gb/s (Total 4 slaves: 48 cores, 96 GB RAM, 2 TB HD) | 4 slave nodes, Node Motherboard: Two Intel Xeon Six core 3.06 GHz, 12 MB cache, 1 HD 7200 RPM, 500 GB, 24 GB RAM, InfiniBand card 40 Gb/s (Total 4 slaves: 48 cores, 96 GB RAM, 2 TB HD) |

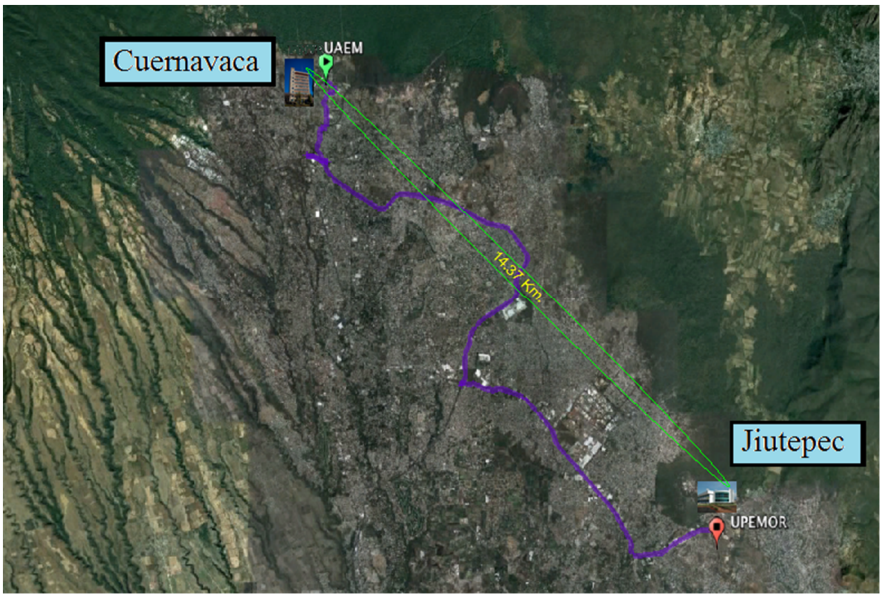


**Figure 3.** Location of MiniGrid in the Mexican state of Morelos.

## 4. Grid-Based Genetic Algorithm to Solve the VRPTW

This paper proposes a grid version of the GA-VRPTW algorithm, named GGA, where the mutation-s operator is applied only to a segment of each population generated in the processes created by the MiniGrid, and a two-stage communication scheme is defined to the result exchanging into the MiniGrid clusters. First, a procedure to combine the mutated segments between cluster nodes is applied, and then the mutated segments are exchanged between the MiniGrid clusters.

### *4.1. Performance Analysis of the GA-VRPTW Algorithm*

Experimental results of the sequential GA-VRPTW algorithm show that it is a very competitive evolutionary algorithm to solve VRPTW instances, but its mutation operator consumes a lot of computing time, as evidenced by his performance analysis (Figure 4). This performance analysis is based on the running time consumed by the algorithm to solve two Solomon benchmark problems (C101 and C106), using five generations and 100 individuals. Figure 4 shows that 99.96% of the computation time in the sequential algorithm is used for the mutation-s operator. In contrast, the *best-selection* operator uses only 0.04% of the computation time.

Flat profile:
Each sample counts as 0.01 seconds.

| % time | cumulative seconds | self seconds | calls | self s/call | total s/call | name |
|---|---|---|---|---|---|---|
| 99.96 | 136.79 | 136.79 | 5 | 27.36 | 27.36 | mutacion(int, int, int) |
| 0.04 | 136.84 | 0.05 | 5 | 0.01 | 0.01 | seccionbest(int, int, int) |
| 0.01 | 136.85 | 0.01 | 99 | 0.00 | 0.00 | Permutaciones(int *, int) |
| 0.00 | 136.85 | 0.00 | 100 | 0.00 | 0.00 | CALCULARUTAS() |
| 0.00 | 136.85 | 0.00 | 5 | 0.00 | 0.00 | calculadisttotal() |
| 0.00 | 136.85 | 0.00 | 5 | 0.00 | 0.00 | crossover(int, int, int, int, int, int) |
| 0.00 | 136.85 | 0.00 | 1 | 0.00 | 0.00 | global constructors keyed to archive |
| 0.00 | 136.85 | 0.00 | 1 | 0.00 | 0.00 | rutakmeans() |
| 0.00 | 136.85 | 0.00 | 1 | 0.00 | 0.00 | MatDistancias() |
| 0.00 | 136.85 | 0.00 | 1 | 0.00 | 0.00 | genpobinialgo0() |
| 0.00 | 136.85 | 0.00 | 1 | 0.00 | 136.85 | algoritmogeneticoKOKO() |
| 0.00 | 136.85 | 0.00 | 1 | 0.00 | 0.00 | _static_initialization_and_destruction_0(int, int) |
| 0.00 | 136.85 | 0.00 | 1 | 0.00 | 0.00 | std::operator\| (std::_Ios_Openmode, std::_Ios_Openmode) |

**Figure 4.** Performance analysis of the methods in the GA-VRPTW algorithm [1].

Based on these results, the selection and crossover operators are applied to the entire population in each process, but the mutation-s operator modifies only one segment of the population in each MiniGrid process.

### 4.2. Grid-based Genetic Algorithm

GGA splits the population of candidate solutions into several segments, which are mutated in each process of the MiniGrid nodes. Figure 5 shows the general scheme of the algorithm. The population division bounds are defined in Equations (12) and (13).

$$Lower_i = N_i\,[L/NN + N_i < res\ ?\ 1{:}0] + N_i < res\ ?\ 0{:}\ res; \qquad i = 0, \ldots, NN-1 \tag{12}$$

$$Upper_i = Lower_i + L/NN + (N_i < res\ ?\ 1{:}0) - 1; \qquad i = 0, \ldots, NN-1. \tag{13}$$

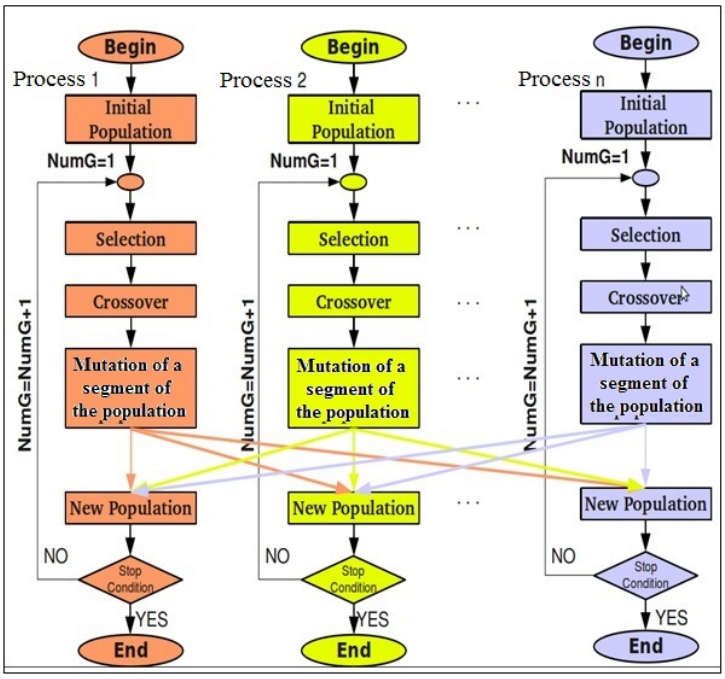


**Figure 5.** General scheme of the grid-based genetic algorithm (GGA) algorithm.

In Equations (12) and (13), $L$ is the population size, $N_i$ is the $i$-th node, $NN$ is the number of nodes in the MiniGrid, and $res$ represents the remainder of the division between $L$ and $NN$.

First, one initial population with only feasible candidate solutions is created in each process of the MiniGrid. Next, these populations evolve until an optimal solution is found, or until a specified number of generations is reached. The genetics operators are applied as follows:

1. Each process applies both the *best-selection* operator and the *crossover-k* operator in the entire population.
2. The resulting population is split into $NN$ segments, using the $Lower_i$ and $Upper_i$ values, and the $i$-th node $N_i$ applies the mutation-s operator in the assigned segment.
3. Each process sends the mutated segment to the other processes of the MiniGrid. All collected chromosomes are used to build a new population.

The distributed mutation step and the segments migration are conducted in each process of the MiniGrid, allowing the increase in the population diversity, as compared to that of the original GA-VRPTW. For example, if the GGA is running in a MiniGrid with 20 process, using a mutation rate of 80%, and each population has 100 candidate solutions, each process applies the selection and crossover operators in the entire population and the mutation operator js applied with 80 individuals. These mutated individuals migrate to the other 19 processes in a balanced way, i.e., four mutated individuals per population.

### 4.3. Distribution and Migration of Mutated Segments

To ensure that the mutated segments are shared between the processes created by the MiniGrid, broadcast with MPI_Scatter function is used to distribute the mutated individuals from each process to the rest of the processes. Each process receives the same amount of sent information so the size of the population will be the same in each process of the MiniGrid. For example, in Figure 6, the MPI_Scatter function distributed the individuals of the population in an equal and parallel manner. The individuals that were stored in an array of structures were distributed to created processes. Each row in the array represents an individual, which is a solution, and a sequence of six customers of the VRPTW. In this example, MPI_Scatter distributed two processes, P1 and P2, in a parallel manner. Individual one was distributed to process P1, and individual two to process P2, and so on. This is done in each process that contains a population of individuals. One process is generally assigned to each core of the MiniGrid, but the MiniGrid can be overloaded so that there is more than one process per core. Each process can only read the population that it owns and can only be responsible for distributing information to each of the remaining processes.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Individual 1 | 1 | 2 | 3 | 4 | 5 | 6 | P1 |
| Individual 2 | 2 | 3 | 6 | 1 | 4 | 5 | P2 |
| Individual 3 | 3 | 2 | 4 | 5 | 6 | 1 | P1 |
| Individual 4 | 6 | 3 | 1 | 4 | 5 | 2 | P2 |
| Individual 5 | 4 | 5 | 1 | 6 | 2 | 3 | P1 |
| Individual 6 | 2 | 6 | 3 | 4 | 1 | 5 | P2 |

**Figure 6.** Distribution of mutated segments in the processes of the MiniGrid.

Figure 7 shows the general scheme of this distribution stage, where S1 is the mutated segment of process 1, s2 is the mutated segment of process 2, and so on. Each process only alters the population assigned to it and, at the end of its mutation stage, must send the mutated segment to the other processes. One process is generally associated with each MiniGrid core, but the environment can be overloaded so that one core can run more than one process.

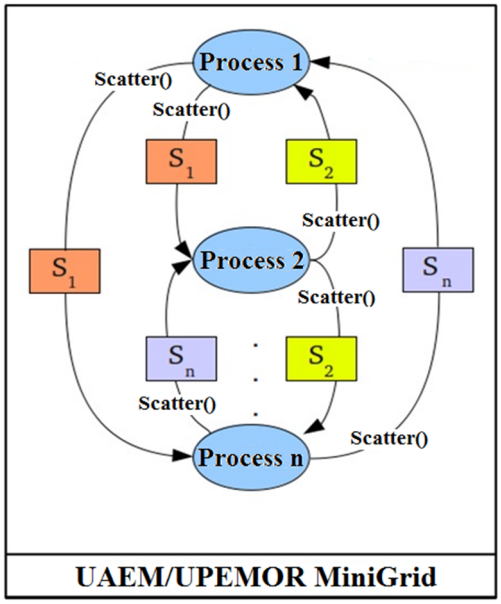


**Figure 7.** Sending mutated segments of the population to all processes in MiniGrid.

Figure 8 shows the final population created after applying the distributed mutation and conducting the segments migration between the MiniGrid processes.

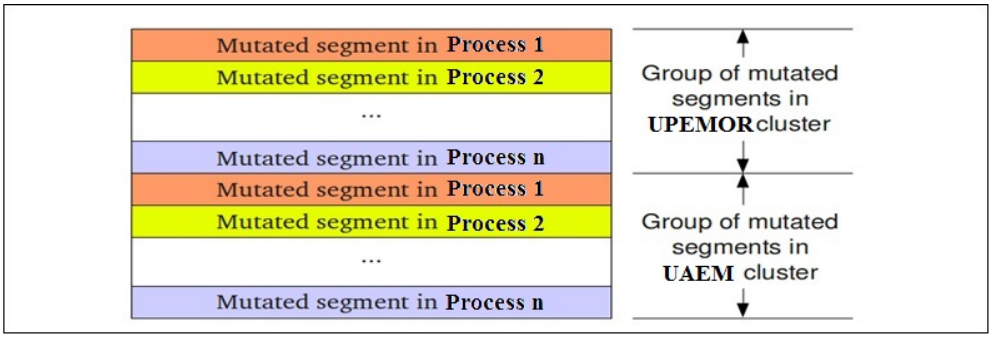


**Figure 8.** Population produced by the combination of mutated segments in the MiniGrid processes.

In GGA, the exploitation rate is high since each MiniGrid process carries out a specialized mutation in conjunction with one iterative local search to improve the quality of the candidate solutions. Furthermore, since a different population evolves in each process by means of one independent recombination, the exploration of new candidate solutions is conducted in different areas of the solution space.

## 5. Experimental Study

In this section, the experimental study carried out to analyze the GGA performance is detailed. A description of the problems used in this study, as well as the definition of the GGA parameters, is given.

### *5.1. Experimental Setup*

The data used to measure the efficiency of GGA in MiniGrid are nine C-type Solomon benchmark problems (C101 to C109) [55], and 10 Gehring and Homberger benchmark problems (C1_10_1 to C1_10_10) [15]. The Solomon benchmark problems have 100 customers, each of whom always places an order and has a maximum of 25 vehicles. Gehring and Homberger benchmark problems use 1000

customers. GGA tests use a population of 100 individuals, the evolutionary process is conducted through 120 generations, the crossover rate is 100%, and the mutation factor is defined using Equations (12) and (13). In this experimental study, GGA is run 30 times, and the average fitness value of these runs is used as the performance value. This value is compared with those obtained to other approaches described in the existing literature.

Furthermore, two non-parametric statistical tests are applied to carried out a statistical analysis of the results produced by the GGA method when comparing them with those obtained by other algorithms. Non-parametric statistical tests are used in this work since it is known that the experimental studies involving evolutionary algorithms do not fulfill the necessary conditions (independency, normality, and homoscedasticity) to apply a parametric test such as ANOVA [56]. In accordance with Derrac et al. [57], two post-hoc tests are carried out: The Wilcoxon test [58] is conducted to compare two algorithms, and the Friedman test [59] and the Bergmann–Hommel post-hoc procedure [60] are used to compare several algorithms.

One non-parametric statistical test evaluates the statistical significance of the average rank of the experimental results through computing the $p$-value without making any assumptions about the distribution of the analyzed data. This $p$-value is used to accept or to reject the null hypothesis of the experiment, which holds that the performance of the compared algorithms does not present significant differences. If the $p$-value does not exceed a predefined significance level (0.05 in this work), the null hypothesis is rejected, indicating that the algorithms are statistically different. Wilcoxon test is designed to compare the two experiments, and two tests must be conducted to compare three or more algorithms: First, the Friedman test is used to detect differences between multiple experiments. Next, if statistical differences exist, a post-hoc test must be conducted to detect the differences between all existing pairs of algorithms. In this work, the Bergmann–Hommel test is used as a post-hoc test since it analyzes the differences of each possible pair of algorithms and whether to reject each of them or not. These statistical tests are applied using the scmamp R library [61].

### 5.2. Methology Applied to Analyze the GGA Performance

Three indicators are evaluated in the experimental study: Latency, speedup, and solution quality. Latency measures the time delay experienced in one distributed environment, and it is analyzed to determine the effect of using the two-stage scheme to migrate information between the nodes of the MiniGrid, and its value is based on the transfer rates between the two clusters in the MiniGrid.

Two processes are used in these experiments: First, 120 processes were used in the MiniGrid (one process for each core), and then an overload of up to 500 uniformly distributed processes was used to evaluate the latency generated by GGA in the MiniGrid. On the other hand, speedup measures the gain achieved by using the parallel program over the sequential version, and its value is computed in Equation (14).

$$speedup = \frac{T_1}{T_n} \tag{14}$$

where $T_1$ is the total running time of the sequential version of the algorithm and $T_n$ is the running time of the parallel version using $n$ processes. The speedup was analyzed to determine the effect of increasing the number of nodes for the distributed application of mutation operator.

Furthermore, to obtain reliable estimates of the results quality, the relative error (RE) is used to compare the results of this experimental study with those produced by other approaches to solve the VRPTW. RE is a factor measuring the difference between the result generated by one experiment and the best-known result value in the existing literature. RE is defined in Equation (15).

$$RE = 100\frac{f(x) - f\left(x^{opt}\right)}{f(x^{opt})} \tag{15}$$

where $f(x)$ is the fitness value result of the current experiment and $f(x^{opt})$ is the best-known fitness value reported in the existent literature.

GGA results are compared with those achieved by the following approaches described in the existing literature:

- GA-VRPTW [1]: Sequential GA-based algorithm with specialized operators to solve VRPTW instances. These operators are used in the GGA. Its experimental study is conducted using a computer with one 1.6 GHz Pentium processor, and the algorithm is implemented in Microsoft Visual C++ 6.0.
- KDMSS [11]: This method is identified in the existing literature using the initials of its authors (Kohl, Desrosiers, Madsen, Solomon, and Soumis). Its experimental study is carried out in one HP9000/829 computer with a 100 MHz PA7200 processor. The KDMSS algorithm is implemented in ANSI C.
- CPLA (cooperative population learning algorithm) [22]: Author uses a cluster HOLK of 256 computers with one Intel Itanium 2 Dual Core. The proposed algorithm has been implemented in Java and uses a software framework to peer-to-peer applications.
- BCO-SIH (a bee colony optimization algorithm with a sequential insertion heuristic) [23]: This algorithm is implemented using Java and performed on one computer with an Intel Core I3 processor.
- MOGA (multi-objective genetic algorithm) [24]: The algorithm is implemented in C/C++ and run on a Silicon Graphics machine with 128 CPUs.
- ACO-TS (ant colony optimization and Tabu search) [25]: The algorithm has been programmed in C++, and the experimental study is carried out in an Intel Core I3 processor machine.
- HM4 (Gehring and Homberger 4) [15]: Experiments are conducted in one computer with a Pentium processor.
- LC03 (Le Bouthillier and Crainic version 03) [17]: The algorithm run in one cluster of five Pentium III computers.
- RP (Ropke and Pisinger) [18]: Experiments are performed on a Pentium IV processor machine, and the heuristic is implemented in C++.

Furthermore, the GGA results with the Gehring and Homberger benchmark problems are compared with the best-known values obtained by other implementations such as Q (Quintiq's optimization technology) [62], CAINIAO (Cainiao network technology) [63], and SCR (Emapa Inc.) [64].

## 6. Results and Discussion

In this section, the experimental results and the statistical tests applied to evaluate these results are outlined. Finally, a discussion about the performance of the GGA method is provided.

### 6.1. Latency Results

Two tests have been performed to measure the latency effects in the GGA performance: The latency rate in a time interval is observed in the first experiment, and the relation between the latency and the number of processes used to run the algorithm is analyzed in the second test. A balanced processes assignation is applied in these tests: 50% of the processes is assigned in the UAEM cluster, and the remaining are assigned to the UPEMOR cluster.

The test to analyze the latency rate in a time interval has been performed with several pairs of nodes selected in the two clusters sending packages of 64 bits. Figure 9 shows the average latency obtained throughout five days, from 9:00 to 16:00. A latency reduction is recorded as the day progresses: Higher latencies are reported at 10:00 (between 57 and 58 msec), and a latency value of less than 52 msec is observed at 15:00. This indicates that in the afternoon, the data transfer is more efficient. Since latency affects the GGA efficiency, it is recommended running the algorithm with it in low (in the afternoon).

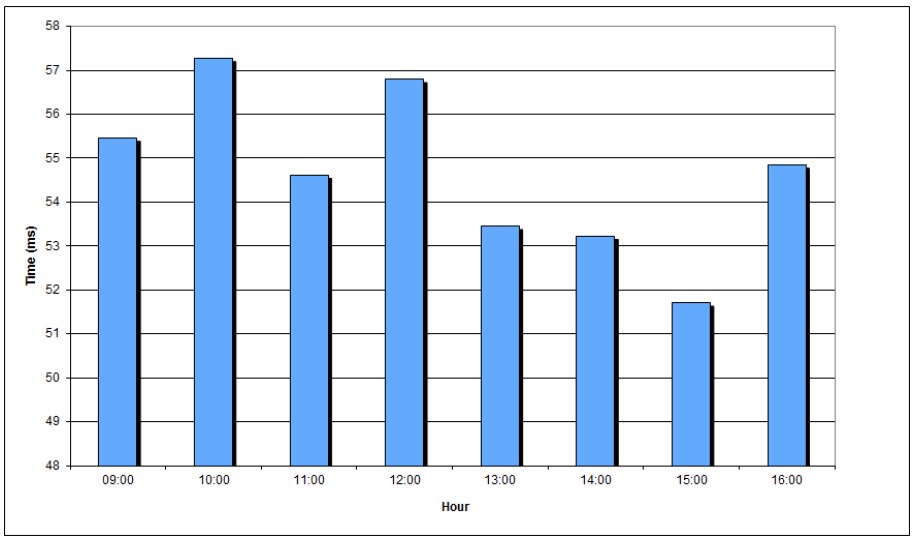


**Figure 9.** Latency rate in a time interval wih the MiniGrid.

Figure 10 shows the latency behavior using a different number of processors assigned in the MiniGrid. It is observed that the latency decreases with increasing MiniGrid processes overload. This behavior is due to the fact that the bandwidth existing in the cluster communication is near to 30 Mbs, meaning that when there are a higher number of processes executed by GGA, more data are sent among clusters. A maximum of 15 Mb of data per second can be sent from UAEM cluster to UPEMOR cluster, and vice-versa, without saturating the microwave communication channel. It is crucial to conduct one proper process balancing to ensure the maximum efficiency of GGA. An overload tends to produce one inefficient running behavior on each cluster.

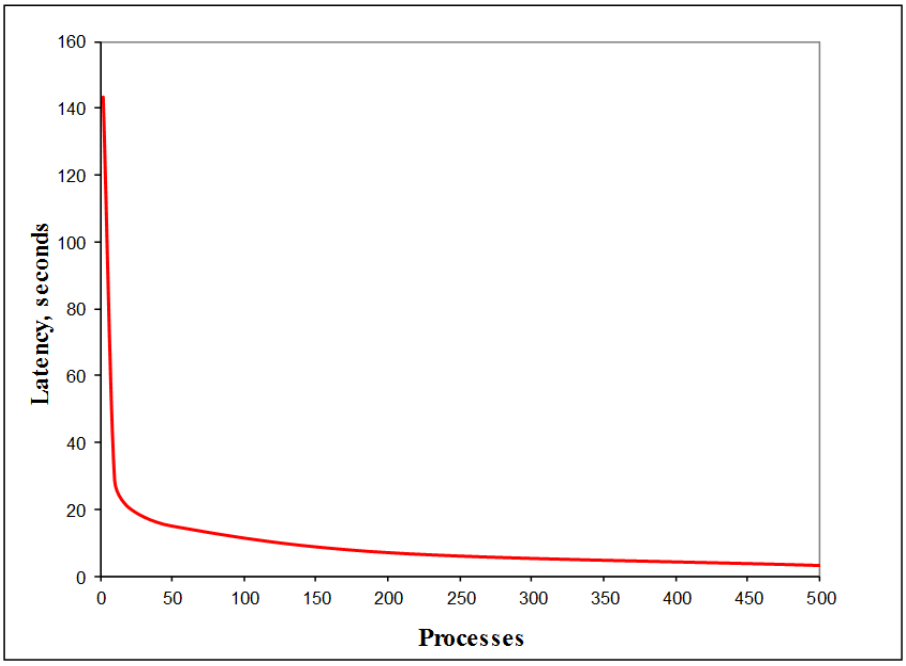


**Figure 10.** The latency rate with different number of processes.

### *6.2. Speedup Results*

To analyze the speedup behavior in the MiniGrid, the GGA results using one cluster are analyzed, since the two clusters in the MiniGrid are homogeneous. The Intel compiler is used with the minimum

and maximum number of processes, so that there is always a maximum of one process (population) per core. Figure 11 shows the speedup observed in the UAEM cluster. The real speedup is very close to ideal and at some points is better than it, for example with 5 and 20 processes. By using the -O3 option for the Intel compiler, the speedup significantly improves. Starting at 20 processes, there is a super-linear speedup in homogeneous parallel machines. It decreases starting at 30 processes but remains a super-linear speedup. Obtaining super-linear speedup for parallel evolutionary algorithms has already been studied in [65] when the algorithm needs a lower number of evaluations in the process communication, which is the case in the proposed GGA.

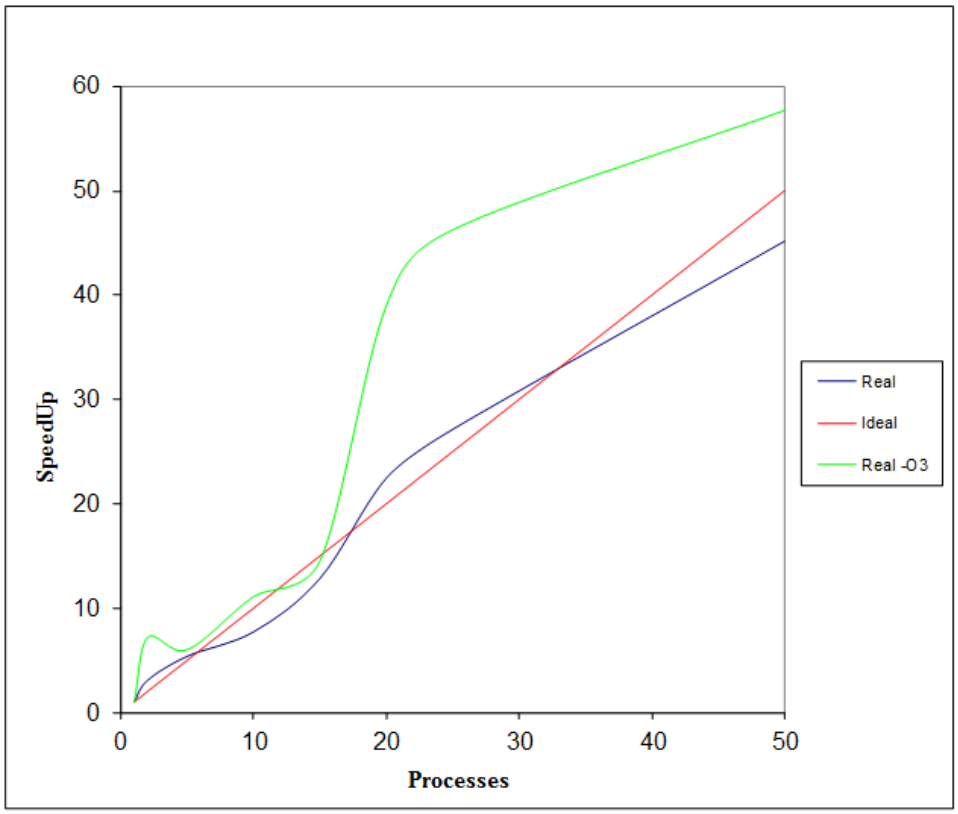


**Figure 11.** Speedup of the GGA in the Autonomous University of Morelos State (UAEM) cluster.

### *6.3. Solutions Quality*

Table 4 presents the experimental results obtained by both GA-VRPTW and GGA with nine Solomon benchmark problems. In this table, NV indicates the number of delivery vehicles used in the optimum solution, Optimum is the optimal solution of the problem, Time is the time in seconds used to reach the reported result, and RE is the relative error previously described. The numbers in parentheses refer to the ranking reached by each method for each problem. The best results were obtained using 60 processes and a single cluster of the MiniGrid. It can be observed that for all instances, GGA is closer to optimal than GA-VRPTW, and the results are reached with relatively short times.

**Table 4.** Results comparison between GA-VRPTW and GGA with Solomon benchmark problems.

| Problem | Optimum | | GA-VRPTW | | GGA | |
|---|---|---|---|---|---|---|
| | NV | Optimum | Time | RE | Time | RE |
| C101 | 10 | 827.3 | 2223 | 0.0119 (2) | 130 | 0.0111 (1) |
| C102 | 10 | 827.3 | 2734 | 0.1141 (2) | 127 | 0.0100 (1) |
| C103 | 10 | 826.3 | 2196 | 0.1434 (2) | 220 | 0.1141 (1) |
| C104 | 10 | 822.9 | 2515 | 0.1374 (2) | 190 | 0.1310 (1) |
| C105 | 10 | 827.3 | 2313 | 0.0020 (2) | 127 | 0.0001 (1) |
| C106 | 10 | 827.3 | 2131 | 0.0224 (2) | 300 | 0.0203 (1) |
| C107 | 10 | 827.3 | 2048 | 0.0707 (2) | 354 | 0.0704 (1) |
| C108 | 10 | 827.3 | 2051 | 0.0041 (2) | 210 | 0.0002 (1) |
| C109 | 10 | 827.3 | 1985 | 0.0453 (2) | 120 | 0.0451 (1) |
| Average ranking | | | | 2 | | 1 |

The Wilcoxon signed ranks test is used in this experiment, and the *p*-value computed is 0.003843, indicating that GGA has one significant improvement over GA-VRPTW.

Table 5 shows the results obtained with the GGA using 10 Gehring and Homberger benchmark problems. In this table, UB is the upper bound best known to the problem (both for the number of vehicles and for the distance traveled), Best and Worse are the best and the worst values found by GGA for each problem, RE is the relative error, and Mean, Median, and Mode are the statistical measures of the results obtained in 30 runs. GGA best results were obtained using 120 processes. The two clusters of the MiniGrid were used, and the MiniGrid had a balanced process distribution.

**Table 5.** Statistical results for 10 Gehring and Homberger benchmark problems.

| Problem | UB | | GGA | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | NV | Value | NV | Best | Worse | Mean | RE | Median | Mode |
| C1_10_1 | 100 | 42,478.95 | 100 | 42,478.95 | 42,550.95 | 42,514.31 | 0 | 42,515.52 | 42,530.01 |
| C1_10_2 | 90 | 42,222.96 | 90 | 42,278.45 | 42,300.37 | 42,286.83 | 0.0571 | 42,284.32 | 42,283.14 |
| C1_10_3 | 90 | 40,101.36 | 90 | 40,207.71 | 40,287.27 | 40,244.82 | 0 | 40,243.31 | 40,251.22 |
| C1_10_4 | 90 | 39,468.60 | 90 | 39,468.60 | 39,502.36 | 39,502.64 | 0 | 39,493.83 | 39,490.70 |
| C1_10_5 | 100 | 42,469.18 | 100 | 42,469.18 | 42,534.18 | 42,502.83 | 0 | 42,499.45 | 42,498.11 |
| C1_10_6 | 99 | 43,830.21 | 100 | 43,832.10 | 43,900.36 | 43,872.31 | 0.0043 | 43,870.90 | 43,870.53 |
| C1_10_7 | 97 | 43,372.03 | 97 | 43,453.92 | 43,468.22 | 43,457.34 | 0.1888 | 43,455.94 | 43,453.32 |
| C1_10_8 | 92 | 42,660.70 | 94 | 41,954.51 | 41,996.55 | 41,984.76 | * | 41,986.47 | 41,973.45 |
| C1_10_9 | 90 | 40,341.06 | 91 | 40,572.31 | 40,602.31 | 40,591.12 | 0.5010 | 40,590.52 | 40,588.44 |
| C1_10_10 | 90 | 39,852.44 | 90 | 39,933.06 | 40,059.01 | 39,970.21 | 0.1781 | 39,968.01 | 39,952.03 |

It can be observed that for half the instances, GGA obtained the UB, and in one case (the C1_10_8 problem), the UB was improved (41,954.51 versus 42,660.70), because a better solution is found, the RE is represented in Table 5 with *. The execution times for GGA were an average of 15,000 to 16,000 s. Table 5 shows that the failure rate to achieve one known result was no higher than 0.501. This value indicates that GGA is very competitive. On the other hand, none of the significant size problems had a mode value equal to the best UB. The most significant difference between the mode value and the UB is 51, and the smallest difference is 0.4. If the median value is equal to the optimum, this indicates that at least half of the 30 tests reached the optimal solution. It is observed that none of the median values of the problems is equal to the UB. The problems that reach results close to optimal, in their mode and median values, were the C1_10_2 problem and the C1_10_7 problem. C1_10_1, C1_10_3, and C1_10_6 problems are the most challenging regarding reaching good results concerning these statistical values.

Table 6 show a comparison of both NV values and the travel costs obtained by the GGA with those of other similar approaches. Table 6 depicted a comparison of the experimental results with the Solomon benchmark problems. The optimum values of the number of vehicles and travel costs listed

in this table are those reached by the KDMSS algorithm. The GGA algorithm is better than PHGA and LC03 algorithms since it has a lower relative error. Concerning the KDMSS algorithm, GGA is competitive since it has a minimal relative error. The smallest relative error (0.0002) is for the C108 problem, and the most significant relative error (0.1310) is with the C104 problem.

**Table 6.** Comparison of the experimental results with nine Solomon benchmark problems.

| Problem | Optimum KDMSS | | GGA | | CPLA | | ACO-TS | | MOGA | | BCO-SIH | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | NV | Cost | NV | RE | NV | RE | NV | RE | NV | RE | NV | RE |
| C101 | 10 | 827.3 | 10 | 0.0111 (1) | 10 | 0.1982 (3.5) | 10 | 0.1982 (3.5) | 10 | 0.1982 (3.5) | 10 | 0.1982 (3.5) |
| C102 | 10 | 827.3 | 10 | 0.1100 (1) | 10 | 0.1982 (3.0) | 10 | 0.1982 (3.0) | 10 | 0.8872 (5.0) | 10 | 0.1982 (3.0) |
| C103 | 10 | 826.3 | 10 | 0.1141 (1) | 10 | 0.2130 (2.5) | 10 | 0.2130 (2.5) | 10 | –(5.0) | 10 | 1.1388 (4.0) |
| C104 | 10 | 822.9 | 10 | 0.1310 (1) | 10 | 0.2285 (2.0) | 10 | 0.3208 (3.0) | 10 | –(5.0) | 10 | 7.5538 (4.0) |
| C105 | 10 | 827.3 | 10 | 0.0001 (1) | 10 | 0.1982 (3.5) | 10 | 0.1982 (3.5) | 10 | 0.1982 (3.5) | 10 | 0.1982 (3.5) |
| C106 | 10 | 827.3 | 10 | 0.0203 (1) | 10 | 0.1982 (3.5) | 10 | 0.1982 (3.5) | 10 | 0.1982 (3.5) | 10 | 0.1982 (3.5) |
| C107 | 10 | 827.3 | 10 | 0.0704 (1) | 10 | 0.1982 (3.5) | 10 | 0.1982 (3.5) | 10 | 0.1982 (3.5) | 10 | 0.1982 (3.5) |
| C108 | 10 | 827.3 | 10 | 0.0002 (1) | 10 | 0.1982 (3.0) | 10 | 0.1982 (3.0) | 10 | 0.1982 (3.0) | 10 | 0.5355 (5.0) |
| C109 | 10 | 827.3 | 10 | 0.0451 (1) | 10 | 0.1982 (3.0) | 10 | 0.1982 (3.0) | 10 | 0.1982 (3.0) | 10 | 1.6149 (5.0) |
| Average ranking | | | | 1 | | 3.06 | | 3.17 | | 3.89 | | 3.89 |

Table 7 shows a comparison of the results with the Gehring and Homberger benchmark problems. Only some algorithms described in the existing literature are compared since they mostly report results for some problems, and it is difficult to make a comparison involving all benchmark instances. Due to this difficulty, in this work, a comparison of the relative error between the GGA solution and the best-known solution is described. GGA is better than the LC03 algorithm since it has a less relative error. Concerning the HM4 algorithm, GGA shows better results in most cases, and the HM4 algorithm is better than GGA in two problems only: C1_10_6 (UB = 42479.15) and C1_10_7 (UB = 42711.39), in which a better upper bound is obtained than the best results reported in the literature (43830.21 and 43372.03), because a better solution is found in HM4, the RE is represented in Table 7 with *. The number of vehicles does not improve, the existing literature reports 99 (Q) and 97 (SCR) vehicles, and the HM4 algorithm reports 100 and 99, respectively. When using GGA, the first problem uses 100 vehicles, and the second uses 97, which are equal to those reported in the literature. GGA find a new upper bound to the C1_10_8 problem (41954.51), but the number of vehicles does not improve since the existing literature reports 92 (SCR) and the GGA achieves 94 vehicles.

**Table 7.** Comparison of the experimental results with 10 Gehring and Homberger benchmark problems.

| Problem | Best Results | | | GGA | | LC03 | | HM4 | |
|---|---|---|---|---|---|---|---|---|---|
| | NV | Cost | Method | NV | RE | NV | RE | NV | RE |
| C1_10_1 | 100 | 42,478.95 | HM4 | 100 | 0.0000 (2) | 100 | 0.0000 (2) | 100 | 0.000 (2) |
| C1_10_2 | 90 | 42,222.96 | CAINIAO | 90 | 0.0571 (1) | 93 | 5.1592 (2) | 93 | 5.2975 (3) |
| C1_10_3 | 90 | 40,101.36 | Q | 90 | 0.0000 (1) | 90 | 9.4338 (2) | 90 | 14.2066 (3) |
| C1_10_4 | 90 | 39,468.60 | Q | 90 | 0.0000 (1) | 90 | 7.3607 (2) | 90 | 12.0356 (3) |
| C1_10_5 | 100 | 42,469.18 | RP | 100 | 0.0000 (1) | 100 | 0.1882 (2) | 100 | 0.18825 (3) |
| C1_10_6 | 99 | 43,830.21 | Q | 100 | 0.0043 (1) | 100 | 3.0824 (2) | 100 | * (3) |
| C1_10_7 | 97 | 43,372.03 | SCR | 97 | 0.1888 (1) | 100 | 1.7596 (2) | 99 | * (3) |
| C1_10_8 | 92 | 42,660.70 | SCR | 94 | * (5) | 97 | 2.7360 (2) | 96 | 1.5263 (1) |
| C1_10_9 | 90 | 40,341.06 | SCR | 91 | 0.5010 (1) | 92 | 11.8351 (2) | 91 | 12.5080 (3) |
| C1_10_10 | 90 | 39,852.44 | SCR | 90 | 0.1781 (1) | 91 | 8.0140 (2) | 90 | 18.6301 (3) |
| Average ranking | | | | | 1.3 | | 2.0 | | 2.7 |

The Friedman test is run with this relative error, and its resulting statistic value is 20.2 for five methods and nine problems, which has a $p$-value of 0.000456. When evaluating this $p$-value with a

significance level of 5%, the null hypothesis is rejected. Next, the Bergmann–Hommel post-hoc test is applied to find all the possible hypotheses that cannot be rejected. In Table 8 is shown both the average rank (AR) of the results yielded by each method and the *p*-values computed by comparing the average accuracies achieved by the GGA versus those obtained by the other methods. The *p*-values highlighted with bold numbers indicate that the null hypothesis is rejected for this pair of methods since they show different performance. Unadjusted *p*-values are calculated with the average ranks of the two methods being compared, as is described by Demšar [66]. These values are used by the Bergmann–Hommel post-hoc test to compute the corresponding adjusted *p*-values. Table 8 shows that the GGA has a better performance than the other methods since it has the lowest average rank (1), and its results are statistically different than the others. Figure 12 shows a graph where the nodes represent the compared methods and the edges joining two nodes indicates that the performance of these methods does not present significant differences. The values shown in the edges are the *p*-values computed by the Bergmann–Hommel post-hoc test. This figure is based on that obtained using the *scmamp* library.

**Table 8.** *p*-values for multiple comparisons among methods.

| Method | AR | GGA *p*-Values | |
|---|---|---|---|
| | | Unadjusted | Bergmann-Hommel |
| CPLA | 3.06 | 0.00581 | 0.02327 |
| ACO-TS | 3.17 | 0.00365 | 0.01460 |
| MOGA | 3.89 | 0.00011 | 0.00106 |
| BCO-SIH | 3.89 | 0.00011 | 0.00106 |
| GGA | 1 | - | - |

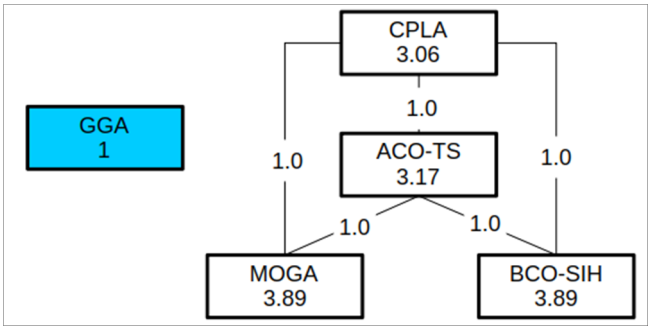


**Figure 12.** *p*-values graph of the compared methods.

Finally, the Friedman statistics computed by analyzing the results produced by these three methods with 10 problems is 9.8, and the corresponding *p*-value is 0.007447, therefore the null hypothesis is rejected. The Bermann–Hommel post-hoc test is then applied to find all possible hypotheses that cannot be refused. Table 9 shows the results of these tests, and Figure 13 shows the graph corresponding to these *p*-values.

**Table 9.** *p*-values for multiple comparisons among methods.

| Method | AR | GGA *p*-Values | |
|---|---|---|---|
| | | Unadjusted | Bergmann–Hommel |
| LC03 | 2 | 0.11752 | 0.11752 |
| HM4 | 2.7 | 0.00174 | 0.00523 |
| GGA | 1.3 | - | - |

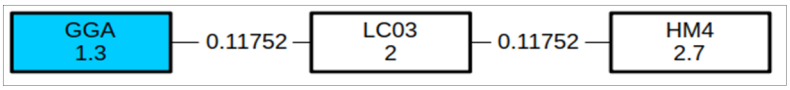


**Figure 13.** *p*-values graph of the compared methods.

The *p*-values obtained by the post-hoc test point out that the GGA method is only statistically different from the HM4 algorithm, and the comparison with the LC03 algorithm indicates that they have a similar statistical performance. However, the GGA has the better average ranking than the LC03 method.

## 7. Conclusions

The results presented in this paper show that the algorithm efficiency improves by increasing the number of processes in the MiniGrid. It is clear that the use of a grid environment to solve complex problems is one useful alternative. The time required to find near-optimal solutions decreases when the cores used in the grid increases, as well as when the communication rate between cores is efficient. The latency in the MiniGrid tends to decline as the day progresses, and it is important to consider the period of the time in which the experimental study is carried out.

By using the collective communication and overload processes for the transmission of mutated segments between the MiniGrid, we can reduce the latency between the clusters geographically distant and connected by microwave. Furthermore, by applying the mutation to a segment of the population in each process executed in parallel, there is a time reduction in creating a new population. Beginning with a random initial generation of individuals in each process, the combination of mutated segments for each process executed in the MiniGrid produces an increase in the exploitation of the search space. The application of the migration scheme between populations increases their diversity, producing good GGA efficacy. GGA performs a low number of evaluations in the communication between processes and with this gets a super-linear speedup.

**Author Contributions:** Conceptualization, M.A.C.-C.; investigation, M.A.C.-C. and M.H.C.-R.; methodology, M.A.C.-C., A.R.-L., and R.R.-L.; validation, M.H.C.-R.; writing—original draft, M.A.C.-C. and R.R.-L.

**Funding:** This research was funded by PRODEP grant number SA-DDI-UAEM/15/451.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Díaz-Parra, O.; Cruz-Chávez, M.A. Evolutionary Algorithm with Intelligent Mutation Operator that Solves the Vehicle Routing Problem of Clustered Classification with Time Windows. *Polish J. Environ. Stud.* **2008**, *17*, 91–95.
2. Christofides, N.; Mingozzi, A.; Toth, P. Exact Algorithms for the Vehicle Routing Problem Based on Spanning Tree and Shortest Path Relaxation. *Math Program.* **1981**, *10*, 255–280. [CrossRef]
3. Christofides, N.; Mingozzi, A.; Toth, P. State Space Relaxation Procedures for the Computation of Bounds to Routing Problems. *Networks* **1981**, *1*, 1145–1164. [CrossRef]
4. Naddef, D.; Rinaldi, G. *Branch and Cut Algorithms for the Capacitated VRP*; SIAM: Philadelphia, PA, USA, 2001; Chapter 3; pp. 53–84. [CrossRef]
5. Gendreau, M.; Hertz, A.; Laporte, G. A Taboo Search Heuristic for the Vehicle Routing Problem. *Manag. Sci.* **1994**, *4*, 1276–1290. [CrossRef]
6. Li, X.; Tian, P. An Ant Colony System for the Open Vehicle Routing Problem. In Proceedings of the ANTS 2006 Conference, LNCS 4150, Brussels, Belgium, 4–7 September 2006; Dorigo, M., Gambardella, L.M., Birattari, M., Martinoli, A., Poli, R., Stützle, T., Eds.; Springer: Berlin, Germany, 2006; pp. 356–363.
7. Brandão de Oliveira, H.C.; Vasconcelos, G.C.; Bastos-Alvarenga, G. Reducing Traveled Distance in the Vehicle Routing Problem with Time Windows Using a Multi-Start Simulated Annealing. In Proceedings of the 2006 International Joint Conference on Neural Networks, Vancouver, BC, Canada, 16–21 July 2006.

8. Rajmohan, M.; Shahabudeen, P. Genetic Algorithm Based Approach for Vehicle Routing Problem with Time Windows. *Int. J. Logist. Syst. Manag.* **2008**, *4*, 338–365. [CrossRef]
9. Bräysy, O. Genetic Algorithms for the Vehicle Routing Problem with Time Windows, special issue on Bioinformatics and Genetic Algorithms. *Arpakannus* **2001**, *1*, 33–38.
10. Cruz-Chávez, M.A.; Díaz-Parra, O.; Juárez-Romero, D.; Martínez-Rangel, M.G. *Memetic Algorithm Based on a Constraint Satisfaction Technique for VRPTW, LNAI 5087*; Springer: Berlin, Germany, 2008; pp. 376–387.
11. Cruz-Chávez, M.A.; Díaz-Parra, O. Evolutionary Algorithm for the Vehicles Routing Problem with Time Windows Based on a Constraint Satisfaction Technique. *Comput. Sist.* **2010**, *13*, 257–272.
12. Kohl, N.; Desrosiers, J.; Madsen, O.B.G.; Solomon, M.M.; Soumis, F. 2-Path Cuts for the Vehicle Routing Problem with Time Windows. *Transp. Sci.* **1999**, *33*, 101–116. [CrossRef]
13. Arbelaitz, O.; Rodriguez, C.; Zamakola, I. Low Cost Parallel Solutions for the VRPTW Optimization Problem. In Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'01), Washington, DC, USA, 3–7 September 2001.
14. Wieczorek, B. Parallel Independent Simulated Annealing Searches to Solve the VRPTW. In *Man-Machine Interactions 2. Advances in Intelligent and Soft Computing*; Czachórski, T., Kozielski, S., Stańczyk, U., Eds.; Springer: Berlin, Germany, 2011; Volume 103.
15. Gehring, H.; Homberger, J. Parallelization of a Two-Phase Metaheuristic for Routing Problems with Time Windows. *J. Heuristics* **2002**, *8*, 251–276. [CrossRef]
16. Berger, J.; Barkaoui, M.A. Parallel Hybrid Genetic Algorithm for the Vehicle Routing Problem with Time Windows. *Comput. Oper. Res.* **2004**, *4*, 2037–2053. [CrossRef]
17. Bouthillier, A.L.; Crainic, T.G. A Cooperative Parallel Meta-Heuristic for the Vehicle Routing Problem with Time Windows. *Comput. Oper. Res.* **2005**, *32*, 1685–1708. [CrossRef]
18. Ropke, S.; Pisinger, D. *A General Heuristic for Vehicle Routing Problems*; Technical Report; University of Copenhagen: Copenhagen, Denmark, 2005.
19. Nalepa, J.; Czech, Z.J. Adaptive Threads Co-operation Schemes in a Parallel Heuristic Algorithm for the Vehicle Routing Problem with Time Windows. *Theor. Appl. Inform.* **2012**, *24*, 191–203. [CrossRef]
20. Nalepa, J.; Blocho, M. Co-operation in the Parallel Memetic Algorithm. *Int. J. Parallel Program.* **2014**, *43*, 812–839. [CrossRef]
21. Baños, R.; Ortega, J.; Consolación, G.; Fernández, A.; De Toro, F. Simulated Annealing-based Parallel Multi-Objective Approach to Vehicle Routing Problem with Time Windows. *Expert Syst. Appl.* **2013**, *40*, 1696–1707.
22. Barbucha, D. A cooperative population learning algorithm for vehicle routing problem with time windows. *Neurocomputing* **2014**, *146*, 210–229. [CrossRef]
23. Jawarneh, S.; Abdullah, S. Sequential Insertion Heuristic with Adaptive Bee Colony Optimisation Algorithm for Vehicle Routing Problem with Time Windows. *PLoS ONE* **2015**, *10*, e0130224. [CrossRef] [PubMed]
24. Pierre, D.M.; Zakaria, N. Stochastic Partially Optimized Cyclic Shift Crossover for Multi-Objective Genetic Algorithms for the Vehicle Routing Problem with Time-Windows. *Appl. Soft Comput.* **2017**, *52*, 863–876. [CrossRef]
25. Bychkov, I.; Batsyn, M. A Hybrid Approach for the Capacitated Vehicle Routing Problem with Time Windows, Optimization Problems and Their Applications. In Proceedings of the 7th International Conference, OPTA 2018, Omsk, Russia, 8–14 July 2018; pp. 6–81.
26. Wang, C.; Mu, D.; Zhao, F.; Southerland, W.J. A Parallel Simulated Annealing Method for the Vehicle Routing Problem with Simultaneous Pickup-delivery and Time Windows. *Comput. Ind. Eng.* **2015**, *83*, 111–122. [CrossRef]
27. Fujisawa, K.; Kojima, M.; Takeda, A.; Yamashita, M. Solving Large Scale Optimization Problems via Grid and Cluster Computing. *J. Oper. Res. Soc. Jpn.* **2004**, *47*, 265–274. [CrossRef]
28. Zunino, I.; Melab, N.; Talbi, E.-G. A Grid-enabled Framework for Exact Optimization Algorithms. In Proceedings of the 21st European Conference on Modelling and Simulation, Prague, Czech Republic, 4–6 June 2007.
29. Rodríguez-León, A.; Cruz-Chávez, M.A.; Rivera-López, R.; Ávila-Melgar, E.Y.; Juárez-Pérez, F.; Cruz-Rosales, M.-H. A Communication Scheme for an Experimental Grid in the Resolution of VRPTW using an Evolutionary Algorithm. In Proceedings of the Electronics, Robotics and Automotive Mechanics Conference, CERMA 2010, Morelos, Mexico, 28 September–1 October 2010; pp. 108–113.

30. Jing, T.; Hiot, M.; Yew, L.; Ong, S. A Parallel Hybrid GA for Combinatorial Optimization Using Grid Technology. In Proceedings of the IEEE Congress on Evolutionary Computation, Canberra, Australia, 8–12 December 2003; pp. 1895–1902.
31. Melab, N.; Cahon, S.; Talbi, E.-G. Grid Computing for Parallel Bioinspired Algorithms. *J. Parallel Distrib. Comput.* **2006**, *66*, 1052–1061. [CrossRef]
32. Luna, F.; Nebro, A.J.; Alba, E.; Durillo, J.J. Solving large-scale real-world telecommunication problems using a grid-based genetic algorithm. *Eng. Optim.* **2008**, *40*, 1067–1084. [CrossRef]
33. Lim, D.; Ong, Y.-S.; Jin, Y.; Sendhoff, B.; Lee, B.-S. Efficient Hierarchical Parallel Genetic Algorithms Using Grid Computing. *Future Gener. Comput. Syst.* **2007**, *23*, 658–670. [CrossRef]
34. Cruz-Chávez, M.A.; Hernández-Báez, I.; Rodriguez-León, A.; Ávila-Melgar, E.Y.; Juárez-Pérez, F.; Martínez-Oropeza, A. *PSAUPMP Application in Grid EELA-2, Status 5: Parallel Simulated Annealing Algorithm for the Weighted Unrelated Parallel Machines Problem*; Second EELA-2 Grid School: Queretaro, Mexico, 2009; Available online: http://applications.eu-eela.eu/app_list.php?l=20 (accessed on 20 May 2010).
35. Cruz-Chávez, M.A.; Rodríguez-León, A.; Ávila-Melgar, E.Y.; Juárez-Pérez, F.; Zavala-Díaz, J.C.; Rivera-López, R. Parallel Hybrid Evolutionary Algorithm in Grid Environment for the Job Shop Scheduling Problem. In Proceedings of the 2nd EELA-2 Conference, Choroni, Venezuela, 25–27 November 2009; pp. 227–234.
36. Escuela, G.; Cardinale, Y.; González, J. A Java-based Distributed Genetic Algorithm Framework. In Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence, Patras, Greece, 29–31 October 2007; pp. 437–441.
37. Garey, M.R.; Johnson, D.S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*; W.H. Freeman and Company: New York, NY, USA, 2003.
38. Toth, P.; Vigo, D. *The Vehicle Routing Problem*; SIAM: Philadelphia, PA, USA, 2002; 367p.
39. Dantzig, G.B.; Ramser, R.H. The Truck Dispatching Problem. *Manag. Sci.* **1959**, *6*, 80–91. [CrossRef]
40. Holland, J.H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*; MIT Press: Cambridge, MA, USA, 1998.
41. Networking and Emerging Optimization Research Group. *Vehicle Routing Problem*; Universidad de Málaga: España, Spain, 2018; Available online: http://neo.lcc.uma.es/vrp/ (accessed on 7 January 2013).
42. Loyd, S.P. Least Squares Quantization in PCM. *IEEE Trans. Inf. Theory* **1982**, *4*, 129–137. [CrossRef]
43. Papadimitriou, C.H.; Steiglitz, K. *Combinatorial Optimization: Algorithms and Complexity*; Dover Publications Inc.: Upper Saddle River, NJ, USA, 1998; p. 496.
44. Lau, K.K.; Kumar, M.J.; Achuthan, N.R. Parallel Implementation of Branch and Bound Algorithm for Solving Vehicle Routing Problem on NOWs. In Proceedings of the 1997 International Symposium on Parallel Architectures, Algorithms and Networks, Taipei, Taiwan, 20 December 1997.
45. Schulze, J.; Torsten, F. A Parallel Algorithm for the Vehicle Routing Problem with Time Window Constraints. *Ann. Oper. Res.* **1997**, *4*, 585–607.
46. Jiang, W.; Zhang, Y.; Xie, J. A Particle Swarm Optimization Algorithm with Crossover for Vehicle Routing Problem with Time Windows. In Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Scheduling, Nashville, TN, USA, 30 March–2 April 2009; pp. 103–106.
47. Cantú-Paz, E. A Survey of Parallel Genetic Algorithms. *Calculateurs Parallèles Réseaux Systèmes Répartis* **1998**, *10*, 141–171.
48. Bote-Lorenzo, M.L.; Dimitriadis, Y.A.; Gómez-Sánchez, E. Grid Characteristics and Uses: A Grid Definition. In *First European across Grids Conference (ACG'03)*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 291–298.
49. Ghosh, S. *Distributed Systems: An Algorithmic Approach*; Chapman & Hall/CRC: London, UK, 2007.
50. Mache, J.; Tyman, D.; Pinter, A.; Allick, C. Performance Implications of Using VPN Technology for Cluster Integration and Grid Computing. In Proceedings of the International Conference on Networking and Services, Slicon Valley, CA, USA, 16–18 July 2006; pp. 75–80.
51. Cordova-Serrano, M.A. Design and Implementation of Connectivity Infrastructure of the Data Center UAEM-UPEMOR Minigrid. Master's Thesis, Polytechnic University of Morelos State (UPEMOR), Jiutepec, Mexico, 2015; p. 198. (In Spanish).
52. Foster, I.; Karonis, N.T. A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems. In Proceedings of the 1998 ACM/IEEE Conference on Supercomputing, Orlando, FL, USA, 7–13 November 1998; pp. 1–11.

53. Kauhaus, C.; Fey, D. Building Mini-Grid Environments with Virtual Private Networks: A Pragmatic Approach. In Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC'06), Bialystok, Poland, 13–17 September 2006; pp. 111–115.
54. Tatezono, M.; Maruyama, N.; Matsuoka, S. Making Wide-Area, Multi-Site MPI Feasible Using XenVM. In Proceedings of the Workshop on Frontiers of High Performance Computing and Networking, LNCS 2006, Sorrento, Italy, 4–7 December 2006; pp. 387–396.
55. Solomon, M.M. *VRPTW Benchmark Problems*; Northeastern University: Boston, MA, USA, 2018. Available online: http://w.cba.neu.edu/~{}msolomon/problems.htm (accessed on 24 March 2005).
56. García, S.; Molina, D.; Lozano, M.; Herrera, F. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 special session on real parameter optimization. *J. Heuristics* **2009**, *4*, 617. [CrossRef]
57. Derrac, J.; García, S.; Molina, D.; Herrera, F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol. Comput.* **2011**, *4*, 3–18. [CrossRef]
58. Wilcoxon, F. Individual Comparisons by Ranking Methods. *Biometrics* **1945**, *4*, 80–83. [CrossRef]
59. Friedman, M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *J. Am. Stat. Assoc.* **1937**, *4*, 675–701. [CrossRef]
60. Bergmann, B.; Hommel, G. Improvements of general multiple test procedures for redundant systems of hypotheses. In *Multiple Hypothesenprüfung/Multiple Hypotheses Testing*; Springer: Berlin, Germany, 1988; pp. 100–115.
61. Calvo, B.; Santafé Rodrigo, G. scmamp: Statistical comparison of multiple algorithms in multiple problems. *R J.* **2016**, *8*, 248–256. [CrossRef]
62. Quintiq. How Can We Prove that Quintiq's Optimization Technology is Number One? Vehicle Routing Problem with Time Windows. Dassault Systèmes. 2014. Available online: https://www.sintef.no/projectweb/top/vrptw/homberger-benchmark/1000-customers (accessed on 10 July 2019).
63. He, Z.; Wang, L.W.; Lin, W.; Chen, Y.; Hu, H. Unpublished Work by CAINIAO AI. 2018. Available online: https://www.sintef.no/projectweb/top/vrptw/homberger-benchmark/1000-customers/ (accessed on 10 July 2019).
64. Cybula, P.; Rogalski, M.; Beling, P.; Jaszkiewicz, A.; Pełka, P. Emapa, S.A. "New Methods of VRP Problem Optimization", Unpublished Research Funded by The National Centre for Research and Development. Project Number: POIR.01.01.01.-00-0222/16. 2018. Available online: https://www.sintef.no/projectweb/top/vrptw/homberger-benchmark/1000-customers/ (accessed on 10 July 2019).
65. Alba, E. Parallel evolutionary algorithms can achieve super-linear performance. *Inf. Process. Lett.* **2002**, *82*, 7–13. [CrossRef]
66. Demšar, J. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.* **2006**, *7*, 1–30.