Comparing multicore implementations of evolutionary meta-heuristics for transportation problems

Raúl Baños*[1,2]
[1]Dpt. of Business Administration and Management, Catholic University of Murcia, Campus de los Jerónimos s/n, E-30107 Guadalupe, Murcia (Spain), (+34)968278656, rbanos@ucam.edu

Julio Ortega[2]
[2]Dpt. of Computer Architecture and Technology, CITIC-UGR, University of Granada, C/Periodista Daniel Saucedo s/n, E-18071 Granada (Spain), jortega@ugr.es

Consolación Gil[3]
[3]Dpt. of Informatics, ceiA3, University of Almería, La Cañada de San Urbano s/n, E-04120 Almería (Spain), cgilm@ual.es

***Abstract*—The set of NP-hard problems require vast computational resources to solve exactly. With the aim of overcoming this limitation several heuristic and meta-heuristic approaches have been proposed in the past. However, the performance of these approaches degradates when solving large problem instances of complex problems. Fortunatelly, parallel processing can be applied to obtain better solutions than the sequential algorithm in the same runtime. The traditional fields of improvement in parallelism have been orientated to experimentation on high-budget equipment, such as clusters of computers or shared memory machines thanks to their high-performance and scalability. In recent years, the generalization of multi-core microprocessors in almost all the computing platforms makes it possible to take advantage of parallel processing even for the desktop computer user. This paper analyzes the performance of population-based meta-heuristics using MPI, OpenMP, and hybrid MPI/OpenMP implementations in a workstation having a multi-core processor when solving a vehicle routing problem, one of the major tasks in the context of transportation. The results obtained when using different number of processes/threads show that these parallel implementations produce high quality solutions compared with the sequential algorithm.**

Keywords: Parallel computing, Multi-core processors, MPI, OpenMP, Hybrid MPI/OpenMP, Vehicle Routing Problems.

I. Introduction

Most real optimization problems cannot be solved exactly because they have extremely large and complex search spaces. To overcome this drawback, researchers have proposed a wide set of meta-heuristics (extensions of heuristic algorithms to tackle general problems) for solving optimization problems arising in several domains. However, the continuous increase in the complexity of real applications often involves that meta-heuristics obtain a poor performance when solving large instances of hard problems in a limited runtime. Thanks to the advances in computer hardware it is now possible to reduce the runtime required to obtain solutions of a given quality even in low-cost computers having multi-core processors [5].

Given a parallel architecture, the decisions to be taken before to parallelize the sequential code are to determine which parallel model is more suitable to implement the algorithm, and the software library to be used. According to the literature [10], the parallel models that are often used to determine the implementation strategy are: (i) The master-worker paradigm, where the master process divides the work amongst the workers, who complete the required work and return the result to the master. The master then organizes the received information, being the master processor responsible for synchronizing communications, collecting and distributing data, etc.; (ii) the diffusion, also known as fine-grained paradigm, that considers a conceptual population like the master-worker paradigm, but this population contains only a few individuals; (iii) the island-based paradigm, also termed distributed or coarse-grained paradigm, consists in dividing the entire population of the sequential algorithm into several sub-populations distributed among different processors. These sub-populations or islands evolve, mainly in isolation, by executing all the steps of the algorithm, although it is possible to share information by migrating solutions between islands. The performance of island-based parallelizations is often influenced by two main design parameters: the migration topology and the frequency of these migrations; (iv) hybrid models that combine different implementation strategies.

Some software components have been standarized to implement parallel algorithms [23], [8], [7], [19]. Pthreads [7] is a low level standard for multi-threaded programming that works by dividing a program into subtasks which execution can be executed in parallel. The MPI (Message Passing Interface) [23] is a portable, efficient, and flexible standard specification for the developers and users of message passing libraries. MPI runs on virtually any hardware platform, including those that based on shared, distributed, and hybrid memory architectures, and allows to write parallel programs by providing routines to initiate and configure the message environment as well as managing some of the tasks of the parallelization, such as decomposing and distributing the starting points of search, moments of communication, synchronization of communications, etc. The OpenMP [8] is a portable and scalable model for specifying shared memory parallelism in Fortran and C/C++ programs in platforms ranging from laptop computers to supercomputers. The standard OpenMP allows the multi-threaded execution of a program thanks to the fact that compiler directives exploit loop level parallelism using the well-known fork-and-join execution model. Further, it is also possible to use a hybrid programming model which uses OpenMP for parallelization inside the node and MPI for message passing

between nodes [9]. In the context of multi-core architectures, the question arises whether it might be advantageous or not to use more than one MPI process with multiple threads running on a node so that there is at least some explicit intra-node communication. The Vehicle Routing Problem (VRP) [12] and its multiple variants have been extensively tackled by sequential [6] and parallel [3] meta-heuristics, but no hybrid MPI/OpenMP implementations have been reported for solving large problem instances of VRPTW. Since both, the OpenMP and the MPI paradigms, have different advantages and disadvantages, and as the VRPTW is a problem whose cost functions are relatively easy to compute but the search space is very large, a priori it is not possible to determine which implementation strategy and software library would obtain the best results. This paper analyzes the performance of population-based meta-heuristics using MPI, OpenMP, and hybrid MPI/OpenMP implementations in a workstation having a multi-core processor when solving the vehicle routing problem with time-windows (VRPTW).

The paper is further organized as follows: Section 2 formally describes the VRPTW. Section 3 presents the framework of this research, including a population-based meta-heuristic (MT-SA) which is parallelized using different parallel models and software libraries. Section 4 presents the results of the empirical analysis carried out in a multi-core workstation, while conclusions are drawn in Section 5.

II. The Vehicle Routing Problem with Time Windows

Vehicle Routing Problem (VRPs) have been the subject of intensive research, not only because its high complexity, but also for its considerable economic impact on all logistic systems [1] and its relevance in supply chain operations. The basic VRP [12] and its variants are NP-hard multi-constrained combinatorial optimization problems that consist in providing goods from a supply point to several geographically dispersed demand points by satisfying a set of constraints. Routes are designed to start and end at the depot and the total demand met by any route cannot exceed the vehicle capacity. The customers are placed in diverse geographical locations and have pre-established requirements of goods and service time. The goods can only be supplied once by exactly one vehicle. Thus, the problem is to minimize the total distance travelled by all the vehicles while satisfying these constraints. The Vehicle Routing Problem with Time Windows (VRPTW) [13] extends the typical Capacitated VRP by including time window constratins, such that customers only allow the service within a given time interval (time window). A large number of approaches have been proposed and discussed in the past, including deterministic [13] and stochastic approaches [6].

The VRPTW is often modeled as a graph theoretical problem [13], where $G = (V, E)$ is a non-directed complete graph, the vertices $V = \{1, ..., N\}$ correspond to the depot and the customers, and the edges $e \in E\{(i, j) : i, j \in V\}$ to the links between them.

Decision variable

$$X_{ij}^k = \begin{cases} 1 & \text{if vehicle } k \text{ travels from node } i \text{ to node } j \\ 0 & \text{otherwise} \end{cases}$$

Parameters

a_j is the earliest time for customer *j* to allow the service,
b_j is the latest time for customer *j* to allow the service,
C_{ij} is the cost of travelling from node *i* to node *j* (here, C_{ij} is considered as the distance or time required for travelling from node *i* to node *j*),
d_j is the demand at customer *j*,
K is the maximum number of vehicles that can be used,
N is the number of customers plus the depot (the depot is noted with number 1, and the customers are noted as 2,...*N*),
Q^k is the loading capacity of vehicle *k*.

Objective function (minimize):

$$TD = \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{N} X_{ij}^k C_{ij} \tag{1}$$

subject to:

$$X_{ii}^k = 0 \qquad (\forall i \in \{1, ..., N\},\ \forall k \in \{1, ..., K\}) \tag{2}$$

$$X_{ij}^k \in \{0, 1\} \qquad (\forall i,j \in \{1, ..., N\}, \forall k \in \{1, ..., K\}) \tag{3}$$

$$\sum_{k=1}^{K} \sum_{i=1}^{N} X_{ij}^k = 1 \qquad (\forall j \in \{2, .., N\}) \tag{4}$$

$$\sum_{i=1}^{N} \sum_{j=2}^{N} X_{ij}^k d_j \leq Q^k \qquad (\forall k \in \{1, .., K\}) \tag{5}$$

$$\sum_{k=1}^{K} \sum_{j=2}^{N} X_{1j}^k \leq K \tag{6}$$

$$\sum_{j=2}^{N} X_{1j}^k - \sum_{j=2}^{N} X_{j1}^k = 0 \qquad (\forall k \in \{1, ..., K\}) \tag{7}$$

$$a_j \leq s_{kj} \leq b_j \qquad (\forall i,j \in \{1, .., N\}, \forall k \in \{1, ..., K\}) \tag{8}$$

$$s_{ki} + C_{ij} - L(1 - X_{ij}^k) \leq s_{kj} \qquad (\forall i,j \in \{1, ..., N\}, \forall k \in \{1, ..., K\}) \tag{9}$$

The objective function is described in Equation (1). Equation (2) denotes that a vehicle must travel from one node to a different one. Equation (3) indicates that a route between two customers can or cannot be covered by a vehicle. Equation (4) states that a customer is visited once by exactly one vehicle. Equation (5) ensures that the goods supplied by a given vehicle k cannot exceed its capacity Q^k. Equation (6) specifies that there are up to K routes going out of the delivery depot, while Equation (7) guarantees that the vehicles depart from and return to the depot. Let s_{kj} be the sum of the distances travelled by vehicle k before arriving at customer j, Equation (8) ensures that time windows are observed. Given a large scalar, L, the inequality represented in Equation (9) specifies that, if vehicle k is travelling from customer i to customer j, the vehicle cannot arrive at customer j before $s_{ki} + C_{ij}$. As

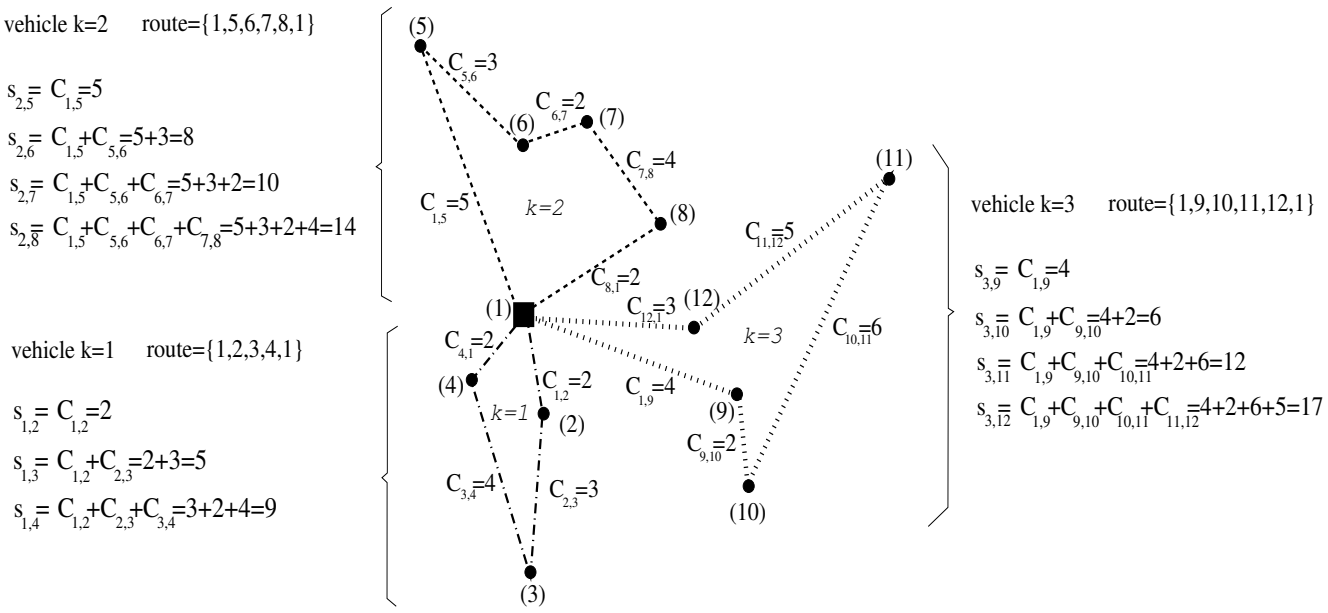

Fig. 1: Determining s_{ki} from a sample solution.

specified by [13], variable s_{kj} corresponds to the time vehicle *k* starts to service customer *j*. If vehicle *k* does not provide any service *j*, s_{kj} is not calculated. With the aim of clarifying this graph-based formulation, Figure 1 provides an example of how the s_{kj} values are calculated for a problem having twelve customers which demand is supplied by three vehicles.

III. ALGORITHMS

A. The Multi-temperature Simulated Annealing (MT-SA)

The Multi-Temperature Simulated Annealing (MT-SA) [2] is population-based algorithm that uses mutation operators to vary the individuals of the population, and Simulated Annealing [17] as selection criterion for each individual of the population. Results obtained by MT-SA in single- and multi-objective contexts [2], [3] show its good performance when solving vehicle routing problems.

MT-SA manages the population of solutions *P* using an integer representation. *P* consists of *p* individuals (solutions), *P*=$\{I_1, I_2, \ldots, I_p\}$, where each individual represents the routes travelled by *K* vehicles to deliver all the customers. Thus, each individual, I_i, is represented by a set of chromosomes, C_{ik}, which consists of a variable number of genes, $C_{ik} = \{1, G^1_{ik}, G^2_{ik}, ..., G^l_{ik}, 1\}$ representing the route of the *k*-th vehicle in the *i*-th individual ($2 \leq G^j_{ik} \leq N$). For example, chromosome $C_{2,4} = \{1, 127, 9, 23, 85, 1\}$ indicates that the fourth vehicle of the second individual departs from the depot and visits customers 127, 9, 23, and 85, before returning to the depot, which is represented by identifier 1. The first and last genes are necessary to verify the constraint described in Equation 7.

The initial routes are built by assigning customers to vehicles until all the former are visited by the latter, such that the constraints are fulfilled. The individuals are optimized by applying ten variation operators often used in this context [13], [24]. Some of them are based on choosing one customer and reallocating it in a different visiting order of the same vehicle (the so called Customer random reallocation operator, and the Customer best reallocation operator), other operators modify the vehicle assigned to the customers (Customer random migration, Customer best migration, Customer random exchange, Customer best exchange, Customer exchange with similar time-window), while other operators divide (Route partition), create (New route), or remove (Route elimination) a given route. When applying variation operators, MT-SA accepts or rejects offspring individuals according to the Metropolis criterion [20] often used by Simulated Annealing (SA) [17]. SA optimizes a solution by exposing it to a high initial temperature, Ti, cooling it by means of a cooling rate, $T_{cooling}$, until the temperature falls below a given threshold, T_{stop}. Therefore, better neighbouring solutions are always accepted, whereas worse solutions are accepted with a certain probability, which is dependent on the current temperature, *t* (when *t* diminishes, the probability of accepting worse solutions decreases). Our approach considers an interval of initial temperatures $[Ti_{min}, Ti_{max}]$, so that the initial temperature of individual I_1 is Ti_{min}, while individual I_p starts in Ti_{max}, and the others are equally distributed along this interval.

B. Parallelization of MT-SA

Since VRPs are very complex problems, there is an increasing interest on the design of parallel strategies for solving them [6]. Several parallel algorithms have been implemented for solving VRPs [18], including some approaches that have applied parallel simulated annealing using clustered SMP architectures by using OpenMP and MPI [11]. The goal of the parallel implementations presented here is to obtain solutions of a higher quality than the sequential algorithms and also to obtain higher quality solutions than the sequential versions without increasing the runtime required by the latter. With the aim of implementing parallel algorithms that present the same characteristics of the sequential code (a parallel simulation of

the sequential code), both the master-worker and the island implementations have been implemented using synchronous communications, i.e. asynchronous message passing (MPI) and the nowait clause (OpenMP) have not been considered. Both paradigms and their hybridization have been adapted to our problem in the following way:

- *Master-worker paradigm with OpenMP*: the master thread initializes the population of solutions (each one containing a valid set of routes to visit all the customers satisfying the constraints), and, in each iteration, the master thread distributes the *p* individuals of population *P* into the number of threads executed (*NTH*), including itself, so that each thread is in charge of optimizing *p*/*NTH* individuals according to the variation operators and the Metropolis function. Once the worker threads have computed their assigned solutions, they return them to the master thread, which computes which is the global best solution, replaces a given percentage of individuals with that solution, and distributes again the work among all the available threads. The master thread is also responsible of controlling the termination condition.
- *Island paradigm with MPI*: each process initializes and optimizes *p*/*NP* individuals autonomously, where *NP* is the number of processes (islands). Periodically, the best solution of each island is sent to a central process which, temporally, is responsible of determining the global best solution and distributing it between the remaining islands. These islands are responsible for copying the received solution in a given percentage of solutions of the population, after which they continue the search process. When the termination condition is fulfilled, all the islands send the solutions to the central process, which returns the global best solution.
- *Hybrid MPI/OpenMP implementation*: the general framework of the hybrid MPI/OpenMP implementation is based on dividing the entire initial population into a set of islands, then applying a parallel scheme based on the master-worker implemented with OpenMP to apply the search operators to the individuals of the island, while MPI is used to establish the communications between the islands by means of message passing. Figure 2 is graphically summarizes the hybrid model in case of having NP=4 processes/islands, and NTH=2 threads per island.

IV. Empirical Analysis

A. Parameters

The parallel computer used in our empirical study is a workstation with a single Intel Core 2 Quad Processor Q6600 (4 cores, 2.40 GHz, 1066 MHz front-side bus, 8MB Cache, 4 GB RAM). The sequential algorithm, coded in C++, has been parallelized using MPI (MPICH2 version 1.2.1p1), and OpenMP (version 3.1). The performance evaluation of the implemented algorithms is analyzed using some of the Gehring & Homberger test problems [14] having 200, 400, and 600 customers have been considered (see below the first column of the tables).

The sequential algorithm uses a population of 160 individuals ($|P|$=p=160),which are initialized using the three heuristics described above. These individuals have a particular annealing scheduling, so that an initial interval of temperatures *Ti*=[1,100] and a slow cooling rate ($T_{cooling}$=0.995) is considered. If the termination condition is not fulfilled and the current temperature falls below T_{stop} (T_{stop}=0.001), the temperature is reinitialized (*t*=*Ti*) and the search process continues. The probability of applying a mutation operator is 25%. If mutation is applied, each of the ten mutation variants is applied with a probability that oscillates between 5% and 15%. When processes or threads communicate to share their best found solutions, the best one is copied in the 25% of the solutions of the population (master-worker paradigm) or of each island (island paradigm).

With the aim of analyzing the advantages provided by the multi-core processor, the parallel implementations using the master-worker model with OpenMP, the island paradigm with MPI, and the hybrid MPI/OpenMP implementations are compared in a single processor using several versions having different number of processes and threads. When comparing different algorithms or implementations, it is possible to determine that one technique is better than another one if it obtains a better performance at a given amount of computational effort. Many authors have commonly proposed the to establish a given number of iterations or fitness evaluations as termination criterion, but it supposes to assume that the cost of other operations is not significant, which is not true in real applications such as VRPTW. This is why in this paper the computational cost is measured in terms of runtime. A total of 15 independent runs with each of these configurations are executed, then analyzing the statistical results obtained.

B. Experimental Results

Table I shows the results obtained by each implementation when establishing a runtime of 60 seconds as termination criterion. Although the best known solutions for those benchmarks have been obtained by other approaches using runtimes of hours or days, to consider a short runtime (60 seconds) is enough to compare different implementations of the same meta-heuristic. Columns 2 to 10 in Table 1 provide the results obtained by each implementation. Each column is marked by two numbers: NP/NTH, where NP indicates the number of processes, and NTH the number of threads. Therefore, column 1/1 denotes the sequential algorithm (MT-SA), columns having values of NP or NTH equal or higher than 2 denote executions of MPI or OpenMP (OMP), respectively, while hybrid MPI/OpenMP implementations correspond to those columns where both, the value of NP and NTH, are equal or higher than 2. The results displayed correspond to the median fitness obtained by each configuration in 15 independent runs per benchmark. It is observed as, given a fixed number of processes (NP), the use of additional threads increases the quality of the results, but not when the product NP*NTH>4, i.e. a performance degradation is observed in the presence of oversubscription [15]. Similarly, a fixed number of threads (NTH), the use of additional processes increases the quality of the results, but not when the product NP*NTH>4. On overall, the results are obtained by the configuration that uses NP=4 processes and NTH=1 thread, i.e. the parallel implementation that considers the island model with MPI using a number

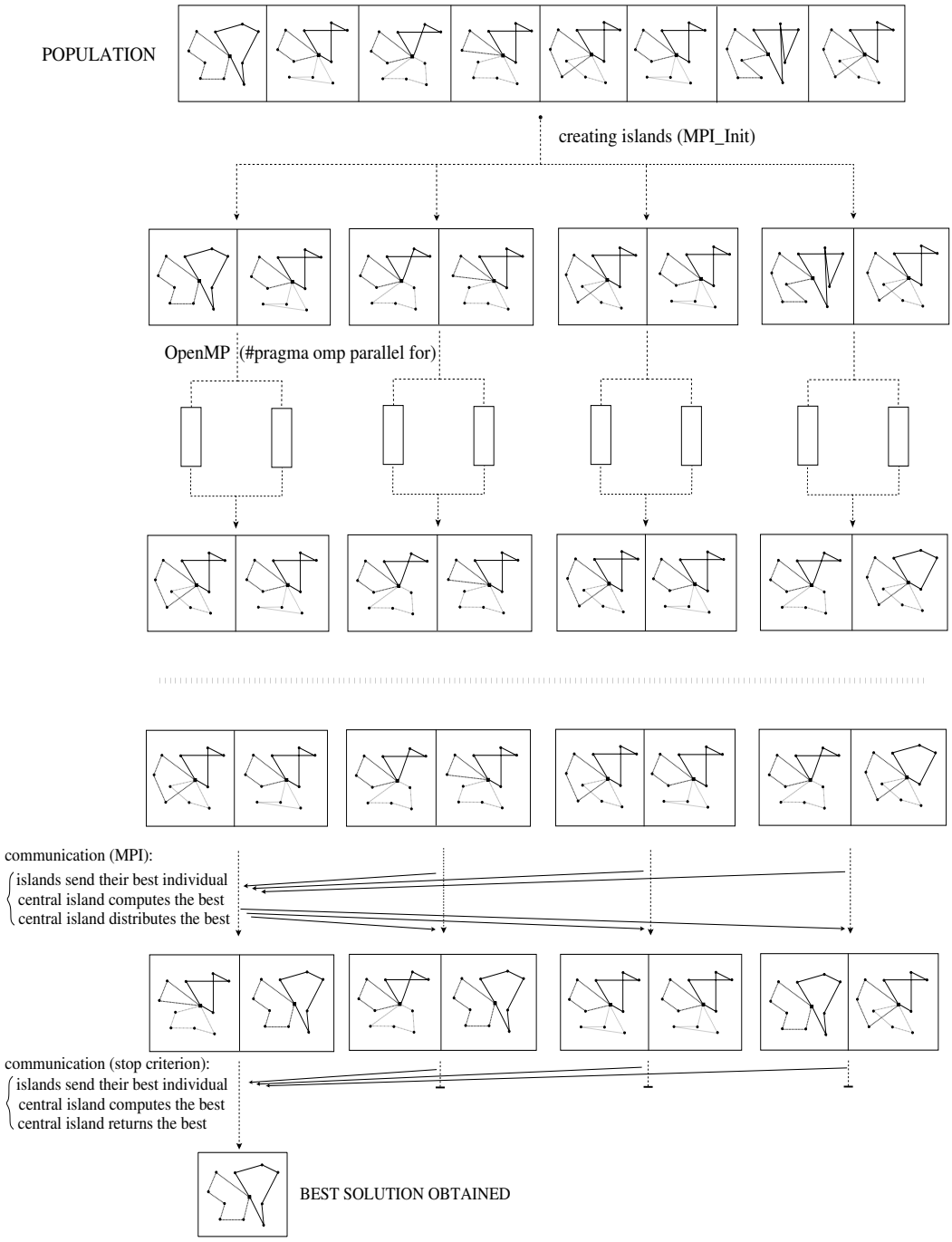

Fig. 2: Graphical description of the hybrid MPI/OpenMP parallelization.

of processes (islands) equal to the number of physical cores available, are slightly better than those obtained by the other parallel versions, while hybrid MPI/OpenMP implementations also obtain good solutions. It can be seen that, when using two processes, the hybrid MPI/OpenMP implementations (columns 7 and 8 of Table I) outperform to the results obtained by the implementation that only uses MPI (column 6). However, it is seen that the best results are obtained by the configuration NP=4/NTH=1, i.e. a pure MPI implementation. The reason arises from the fact that the population size of each island when using NP=4 is smaller (40 individuals per island) than in case of using NP=1 (160 individuals) or NP=2 (80 individuals). This involves that the former configuration is able to perform more iterations within the same runtime (higher intensification), which leads to the rapid convergence of the parallel algorithm.

Whenever several experiments are performed it is important to determine whether or not the variation in the results is significant, i.e. the observed spread of mean values that would not normally arise from the chance variation within groups. With the aim of determining if there is a significant difference between these groups of results obtained in the experiments an

TABLE I: Comparing MT-SA and pMT-SA using different number of processes/islands and threads (15 executions per parallel version and benchmark).

version	Serial	OMP	OMP	OMP	MPI	Hybrid	Hybrid	MPI	Hybrid	(ANOVA)
NP/NTH	1/1	1/2	1/4	1/8	2/1	2/2	2/4	4/1	4/2	p-critical
R1_2_3	8545.37	8058.08	7814.85	7880.74	7562.26	7150.40	7205.50	**6955.25**	7055.95	9.52E-62
R1_2_8	7972.26	7411.90	7272.33	7313.67	7066.43	6657.84	6685.94	**6411.80**	6507.33	1.20E-61
R2_2_3	8414.32	7882.46	7739.64	7776.39	7526.37	7226.28	7293.38	**6898.30**	7019.86	5.31E-55
R2_2_8	8405.96	7809.67	7530.40	7467.80	7374.60	6896.41	6937.61	6642.51	**6618.69**	3.95E-59
C1_2_3	8518.33	7976.75	7671.98	7759.77	7431.71	7016.04	7142.79	**6724.82**	6825.20	4.01E-56
C1_2_8	7226.24	6830.63	6593.76	6654.88	6451.87	6154.73	6165.89	5945.55	**5923.03**	1.92E-48
C2_2_3	7853.91	7459.69	7032.65	7215.11	7002.92	6633.57	6593.90	**5988.00**	6342.65	3.49E-55
C2_2_8	2408.72	2373.98	2368.83	2363.87	2347.29	2326.20	2303.37	**2233.06**	2252.59	4.00E-21
R1_4_3	24148.31	23479.45	23102.62	23127.13	23527.81	22726.62	22835.59	**21940.21**	22557.47	4.38E-41
R1_4_8	25440.55	24687.79	24275.50	24259.41	24359.92	24011.24	23962.57	**23663.68**	23693.81	8.98E-49
R2_4_3	23701.78	23235.58	22774.32	22769.57	22663.70	22342.52	22213.23	**21940.21**	22059.07	2.90E-26
R2_4_8	26845.46	26275.67	25622.80	25760.39	25356.72	25181.23	24813.58	24878.75	**24783.61**	5.89E-33
C1_4_3	26536.13	26022.09	25632.92	25440.84	25922.48	25447.81	25450.36	**25087.54**	25223.98	1.87E-48
C1_4_8	20695.45	20452.01	20241.49	20252.41	20545.15	20227.58	20275.51	**20023.83**	20103.95	5.31E-28
C2_4_3	23702.81	23163.89	22716.10	22616.76	22899.04	22407.35	22384.43	**21894.37**	22202.47	4.91E-35
C2_4_8	8204.18	8083.89	7997.70	7970.28	8135.00	8098.51	8048.97	**7954.78**	7969.40	2.92E-14
R1_6_3	51705.89	51319.97	50862.12	50845.90	51218.85	50649.75	50666.40	50518.97	**50451.73**	4.13E-26
R1_6_8	59238.32	58792.84	58301.67	58228.27	58639.93	58208.97	58162.19	57706.28	**57628.25**	1.71E-21
R2_6_3	57815.29	56930.08	56526.83	56054.73	56190.02	55975.89	56072.31	**55250.38**	55385.22	2.09E-07
R2_6_8	67900.86	66383.01	65977.14	65290.33	66451.94	65127.31	65586.43	**64309.57**	65002.45	2.93E-22
C1_6_3	47374.99	47012.68	46756.57	46796.16	47081.30	46486.99	46558.18	46267.20	**46183.19**	3.43E-28
C1_6_8	34324.97	34212.52	34180.70	34168.41	34079.43	33993.98	34013.65	**33940.60**	34007.59	5.93E-22
C2_6_3	52830.02	51959.49	51512.67	51509.26	51948.42	51178.83	50963.05	50728.43	**50666.69**	4.44E-34
C2_6_8	15786.09	15721.31	15748.39	15698.85	15716.41	15657.65	15642.66	**15604.28**	15657.61	9.93E-06

one-way ANOVA test is applied. Given the typical confidence level of 95%, the null hypothesis is rejected if the probability value (p-value) is smaller than or equal to the critical value (p-critical=0.05). The last column of Table I shows that p-value$\leq$0.05, i.e. the null hypothesis is rejected in all cases, which denotes the existence of a significant variation between the results of the different groups.

Taking into account the previous results, it is now analyzed how the parallel implementations are able to reduce the runtime required to obtain a solution of equal or better quality than the median result obtained by MT-SA after executing 15 independent runs during 60 seconds. Three configurations are analyzed: an OpenMP implementation ($\{$NP=1,NTH=4$\}$), a MPI implementation $\{$NP=4,NTH=1$\}$, and a hybrid OpenMP/MPI implementation $\{$NP=2,NTH=2$\}$. Table II shows the mean, and average deviation of the 15 independent runs carried out by these configurations of pMT-SA. Figure 3 shows the speedup obtained using all the benchmarks using the information contained in Table II. Black bars correspond to the OpenMP implementations, white bars are related to MPI, while grey ones describe the results of the hybrid MPI/OpenMP implementations. These results show that the parallel implementations in all cases need less than 60 seconds to reach a solution of at least the same quality than that obtained by the serial algorithm, i.e. the parallel algorithms obtain an improvement in terms of speedup. At first sight, yielding speedups of $\sim$2 could be considered poor in terms of scalability, but it should be considered that MT-SA is a stochastic approach and, therefore, it is possible that some threads do not perform a given instruction (e.g.: if the conditional expression of a while structure results as false when computed on its own data), and therefore this stream processor is simply put into idle mode during the remaining loops performed by the others. This phenomenon, known as thread divergence [21], often causes serious performance degradation. It is noticeable that the implementations using OpenMP often converge faster that the MPI-based implementations, while MPI outperformed to pure OpenMP implementations when establishing a fixed runtime (see Table I). Here again, the reason if this behaviour seems to be the different degrees of intensification/diversification of MPI/OpenMP implementations.

V. Conclusions and future work

The design of efficient methods for solving vehicle routing problems has become an area of research that has attracted much attention due to its influence in transportation, logistics, and supply chain management. Since this problem is NP-hard, most algorithms presented to solve this problem are based on heuristic and meta-heuristic approaches. Nevertheless, whenever the number of customers is very large, it is necessary to apply techniques, such as parallel processing, to improve the efficiency of these heuristics. Thanks to the generalized use of multi-core processors from supercomputers to laptops, parallel programming has become popular in the scientific community. This paper analyzes the advantages provided by multi-core processors to obtain good quality solutions to the vehicle routing problem with time windows. With this aim, a population-based meta-heuristic based on Simulated Annealing has been parallelized using MPI, OpenMP, and hybrid MPI/OpenMP schemes. Results obtained in a workstation with a single multi-core processor show that these parallel implementations improve the quality of the solutions obtained by the sequential algorithm. Moreover, some theoretical phenomena, such as oversubscription and thread divergence have been observed in our experiments. As future work, it is planned to analyze the behavior of these hybrid parallel algorithms in a cluster, and also to apply them to solve multi-objective formulations of this problem dealing with several objectives and constraints.

TABLE II: Runtime (seconds) required by pMT-SA to obtain a solution of similar quality than that obtained by the serial MT-SA with a runtime of 60 seconds.

	1/4 (OMP)		2/2 (Hybrid)		4/1 (MPI)	
	mean	avg. dev.	mean	avg. dev.	mean	avg. dev.
R1_2_3	30.91	3.99	**25.17**	1.87	30.22	4.36
R1_2_8	34.54	3.29	**29.49**	3.80	40.34	4.95
R2_2_3	31.34	4.10	**24.61**	4.00	31.62	7.29
R2_2_8	34.06	2.90	**29.07**	4.40	36.03	4.62
C1_2_3	29.97	4.75	**29.15**	3.05	38.27	5.29
C1_2_8	35.61	5.94	**30.11**	4.13	42.88	9.54
C2_2_3	35.17	4.58	**31.56**	2.78	40.20	5.65
C2_2_8	**38.57**	8.39	40.10	8.53	45.58	7.26
R1_4_3	**34.30**	4.54	42.34	7.79	46.48	5.85
R1_4_8	**29.52**	3.49	38.52	8.09	40.67	6.33
R2_4_3	**33.96**	6.83	40.65	7.30	44.00	9.10
R2_4_8	**41.73**	5.59	49.08	3.93	47.94	3.19
C1_4_3	**32.75**	4.11	47.83	4.14	42.75	8.08
C1_4_8	**34.96**	7.02	47.26	4.84	43.81	6.32
C2_4_3	37.30	3.45	45.64	5.95	**36.25**	4.39
C2_4_8	**42.87**	7.10	47.39	3.13	45.49	9.34
R1_6_3	**29.89**	8.57	39.87	4.32	33.38	4.92
R1_6_8	**39.31**	7.61	45.54	5.84	40.18	7.36
R2_6_3	**39.60**	8.67	44.72	4.41	44.23	6.15
R2_6_8	**31.72**	6.18	39.73	5.74	45.25	4.86
C1_6_3	**30.32**	5.45	38.23	5.01	43.35	4.46
C1_6_8	**24.74**	9.20	33.69	8.08	30.06	7.84
C2_6_3	**28.41**	4.80	31.68	6.01	29.79	7.59
C2_6_8	39.10	9.51	**32.28**	8.06	40.98	8.10

The application of computational optimization methods for solving complex scientific and industrial applications requires the perform a enormous number of calculations. The number of calculations per unit of time is dependent of the frequency of a chipset, and this frequency depends on semiconductor technology as well as processor architecture. During several decades the processors manufacturers have been able to double the number of transistors every two years, but in practice scaling of semiconductor transistors has reached a limit due to the physical properties, which is why it is expected that CPUs will be limited in frequency for several years. Having in mind this fact as well as the success of multi-core and multi-threading technologies, it is supposed that processors manufacturers, such as Intel or AMD, will continue working in these technologies, probably minimizing the power consumption, and increasing the temperature dissipation and number of cores included in the chip, among other aspects. As commented above, an important advantage of having processors with multi-core and multi-threading technologies is the possibility of applying parallel processors even on desktop and laptop computers. Thus, from the perspective of the efficiency it would be desirable to take advantage of these systems to parallelize not only scientific optimization problems, but also any other tasks associated to software applications. Coming back to the problem at hand, the VRPTW, the application of independent and hybrid parallel implementations of MPI, OpenMP, can be extended to the multi-objective case [16], [4], in which several objectives, such as the travelling distance, number of vehicles, load imbalance, route imbalance, etc. are considered.

In the following, several hybrid MPI/OpenMP implementations are briefly commented: i) A first alternative would be to consider a hybrid implementation in which each island would evolve in parallel, then applying the variation operators to optimize the different objectives simultaneously. In this case, OpenMP would be applied to speedup the algorithm calculations within each island, while MPI communications would be occasionally performed to spread better solutions. Finally, a central island would combine its non-dominated front of solutions with those received from the other islands; ii) A second alternative is to consider a hybrid implementation in which each island would evolve in parallel, but only considering a particular objective to optimize, i.e. there is at least one island responsible to optimize each objective. OpenMP would be applied to speedup the algorithm calculations within each island, while MPI communications would be occasionally performed to spread better solutions. Finally, the different non-dominated fronts generated by each island would be composed into a single front that would represent the output of the algorithm; iii) A third alternative is to divide the search space such that each island is responsible of the search in a particular area. For example, since the VRPTW involves using a given number of vehicles, it would be possible to include a new constraint in each island, such that it only considers feasible a given number or interval of vehicles which intrinsically involves that each island explores different areas of the search space. OpenMP would be applied to speedup the algorithm calculations within each island, while MPI communications would be occasionally performed to spread better solutions. Finally, the different non-dominated fronts generated by each island would be composed into a single front that would be returned as the result of the parallel algorithm; iv) It would be also possible to implement hierarchical parallel schemes to address the previous alternative approaches at different levels.

ACKNOWLEDGMENTS

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness and FEDER funds

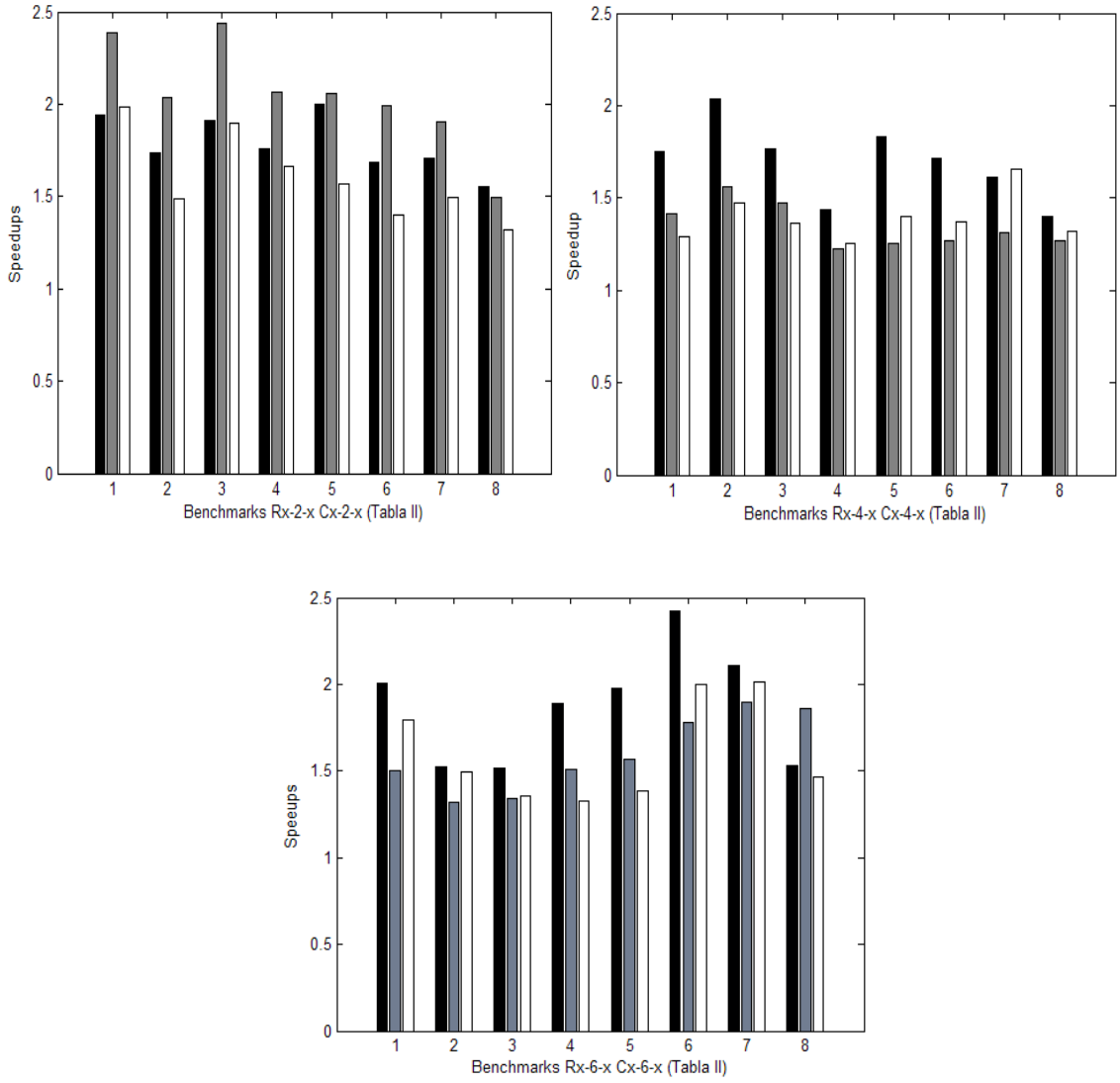

Fig. 3: Speedup according to the benchmark type (OpenMP - black, MPI/OpenMP - grey, MPI - white).

under project TIN2012-32039. R.Baños also acknowledges the support of a Juan de la Cierva postdoctoral fellowship.

REFERENCES

[1] G.B. Alvarenga, G.R. Mateus, G. de Tomic, A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows, Computers & Operations Research 34(6) (2007) 1561-1584.

[2] R. Baños, J. Ortega, C. Gil, A. F. Molina, F. de Toro, A multi-start hybrid algorithm for vehicle routing problems with time windows, World Online Conference on Soft Computing in Industrial Applications, 2011.

[3] R. Baños, J. Ortega, C. Gil, A. Fernández, F. de Toro, A simulated annealing-based parallel multi-objective approach to vehicle routing problems with time windows. Expert Systems with Applications 40(5) (2013) 1696-1707.

[4] R. Baños, J. Ortega, C. Gil, A.L. Márquez, F. de Toro, A hybrid meta-heuristic for multi-objective vehicle routing problems with time windows, Computers & Industrial Engineering 65(2) (2013) 286-296.

[5] G. Blake, R.G. Dreslinski, T. Mudge, A survey of multicore processors, IEEE Signal Processing Magazine 26(6) (2009) 26-37.

[6] O. Bräysy, M. Gendreau, Vehicle routing problem with time windows, part II: Metaheuristics, Transportation Science 39(1) (2005) 119-139.

[7] D.R. Butenhof (1997) Programming with POSIX Threads. Addison-Wesley.

[8] B. Chapman, R. Jost van der Pas, D.J. Kuck (foreword), Using OpenMP: Portable shared memory parallel programming, The MIT Press, 2007.

[9] M.J. Chorley, D.W. Walker, Performance analysis of a hybrid MPI/OpenMP application on multi-core clusters, Journal of Computational Science 1(3) (2010) 168174.

[10] C.A. Coello, G.B. Lamont, D.A. Van Veldhuizen, Evolutionary algorithms for solving multi-objective problems, Genetic and Evolutionary Computation Series, Springer, 2007.

[11] Z.J. Czech, W. Mikanik, R. Skinerowicz, Implementing a parallel simulated annealing algorithm, Springer Lecture Notes in Computer Science 6067 (2009) 146-155.

[12] G.B. Dantzig, J.H. Ramser, The truck dispatching problem, Manage-

ment Science 6(1) (1959) 80-91.

[13] N.A. El-Sherbeny, Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods, Journal of King Saud University (Science) 22(3) (2010) 123-131.

[14] H. Gehring, J. Homberger, A parallel two-phase metaheuristic for routing problems with time windows, Asia-Pacific Journal of Operations Research 18(1) (2001) 35-47. [available at: http://www.sintef.no/Projectweb/TOP/VRPTW/Homberger-benchmark/]

[15] C. Iancu, S. Hofmeyr, Y. Zheng, F. Blagojevi, Oversubscription on multicore processors, IEEE International Parallel and Distributed Processing Symposium, 2010, pp. 1-11.

[16] N. Jozefowiez, F. Semet, E-G. Talbi, Multi-objective vehicle routing problems, European Journal of Operational Research 189(2) (2008) 293-309.

[17] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220(4598) (1983) 671-680.

[18] A. Le Bouthillier, T.G. Crainic, A cooperative parallel meta-heuristic for the vehicle routing problem with time windows, Computers & Operations Research 32(7) (2005) 1685-1708.

[19] A.L. Márquez, C. Gil, R. Baños, J. Gómez, Parallelism on multicore processors using Parallel.FX, Advances in Engineering Software 42(6) (2011) 259-265.

[20] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A. Teller, E. Teller, Equation of state calculations by fast computing machines, The Journal of Chemical Physics 21(6) (1953) 1087-1092.

[21] D. Robilliard, V. Marion, C. Fonlupt, High Performance Genetic Programming on GPU, Proceedings of the 2009 Workshop on Bio-inspired Algorithms for Distributed Systems, 2000, pp. 85-94.

[22] S. Santander-Jimenez, M.A. Vega-Rodriguez, J.A. Gómez-Pulido, J.M. Sánchez-Pérez, Evaluating the Performance of a Parallel Multiobjective Artificial Bee Colony Algorithm for Inferring Phylogenies on Multicore Architectures, Proceedings of the 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications, 2012, pp. 713-720.

[23] M. Snir, S. Otto, S. Huss-Lederman, D. Walter, J. Dongarra, MPI: The complete reference, MIT Press, Boston, 1996.

[24] K.C. Tan, Y.H. Chew, L.H. Lee, A hybrid multiobjective evolutionary algorithm for solving vehicle routing problem with time windows, Computational Optimization and Applications 34(1) (2006) 115-151.