Luna, F., Cervantes, A., Isasi, P. et al. Grid-enabled evolution strategies for large-scale home care crew scheduling. Cluster Comput 21, 1261–1273 (2018)

DOI: https://doi.org/10.1007/s10586-017-1058-2

Cluster Computing manuscript No.
(will be inserted by the editor)

Grid-enabled evolution strategies for large-scale home care crew scheduling

Francisco Luna · Alejandro Cervantes · Pedro Isasi · Juan F. Valenzuela-Valdés

Received: date / Accepted: date

Abstract The Home Care Crew Scheduling (HCCS) problem is a planning task whose goal is to allocate a set of professional caregivers in the most efficient way to perform a number of assistencial and health care visits to the customers private homes. This is part of an important trend in advanced health care systems, to promote "independent living" specially in situations of dependency on long-term care. This not only ensures a higher quality of life but also a lower cost for society. Real instances of the HCCS problem are large and highly constrained due to both caregivers' contract limitations and customers' needs.

This paper presents an advanced parallel model that solves HCCS problems using a grid-based asynchronous evolutionary algorithm. Our approach has been tested using a grid computing facility of up to 300 nodes. The algorithm is a modified $(1+\lambda)$ Evolutionary Algorithm (EA), parallelized using a master/worker model that minimizes communication requirements and processor bottlenecks by distributing both the execution of the EA operators and the evaluation of solutions. We have used three large real-world instances provided by a private company to perform experimentation with different configurations of the EA and number of workers. Results show that our algorithm achieves solutions that clearly outperform the solution provided by the company and the grid-based algorithm is able to handle real world HCCS problems.

Keywords Home care scheduling · parallelism · grid computing · evolutionary algorithms

Francisco Luna
Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga, Málaga (Spain)
E-mail: flv@lcc.uma.es

Alejandro Cervantes
Departamento de Informática
Universidad Carlos III de Madrid, Leganés, Madrid (Spain)
E-mail: alejandro.cervantes@uc3m.es

Pedro Isasi
Departamento de Informática
Universidad Carlos III de Madrid, Leganés, Madrid (Spain)
E-mail: pedro.isasi@uc3m.es

Juan F. Valenzuela-Valés
Departamento Teoría de la Señal, Telemática y Comunicaciones
Universidad de Granada, Granada (Spain)
E-mail: juanvalenzuela@ugr.es

1 Introduction

Management of home care (HC) services for either senior and/or disabled citizens is becoming an endless source of challenging optimization problems for the research community [19,23]. The field of HC services includes delivering both simple health care services and other services such as cooking, cleaning, dressing, bathing, etc., in the place of residence of those citizens, allowing them to stay at home as long as possible and still receive professional help. This business opportunity has driven many different organizations to start delivering these services [4,24]. As many of these services involve taking care of personal needs, customers (or patients) usually prefer being serviced by the same caregiver(s) since the very beginning of the contract. In this way, a relation of familiarity and confidence can be developed. As a consequence, HC service providers usually work by generating a mid-term plan which is only updated due to new or canceled contracts. Small adaptations (e.g., unavailability of a given caregiver) are included as needed in this mid-term plan. It becomes evident that the cornerstone of HC service providers is management of their human resources, so reaching a

cost-efficient plan, subject to a number of operational and logical constraints, becomes a critical issue [13]. This is the problem addressed in this work, which is called the Home Care Crew Scheduling (HCCS) problem.

Different authors propose different formulations of this problem, taking into account specific or theoretical restrictions that may be useful or not in real-world problems [20,10,22]. For instance, caregivers can have transportation requirements, customers can be allowed a set of preferred caregivers, etc. Also dynamic environments, that require daily planning and changing transport times have been considered [25]. It can be considered that most of these are generalized versions of the Vehicle Routing Problem [33], sometimes taking into account uncertainty by using Time Windows [8, 9]. However, HCCS is fundamentally different because of its objective function(s) and more complicated constraints [18]. In [11], HCCS is analysed in the context of the generic framework of workforce scheduling and routing problems. Though parallel architectures are not always considered, certainly real-world routing and crew scheduling problems benefit from high-performance parallel computing resources, either for problem decomposition and independent solving [27] or specially in dynamic environments, when real-time co-adaptation and cooperative learning are required in multi-agent architecture for resource coordination [34].

Instead of proposing a theoretical case, this work has been developed in collaboration with the EULEN company$^{\text{TM}}$to assess an actual real-world scenario. The EULEN group is an international and multidisciplinary service company offering cleaning, health, logistics, environment, security, services, among others, and with presence in 14 countries worldwide. The HCCS problem defined with EULEN has its own features, mainly determined by the human resource policy, the collective bargaining agreement with the trade union, etc., that have several differences with respect to the already published models [18,22]: caregivers are not visiting customers in a tour, all the caregivers can serve any customer (no skills and no preferences are used), and services have a fixed start time (time windows are not used). However, the actual challenging feature of the problem addressed in this work arises from the fact that a single service can be split throughout the week, i.e., Monday from 9:30AM to 11:00AM, Wednesday from 10:30AM to 11:30AM, etc. That is, individual "services" are assigned as "service groups" that must be serviced by the same caregiver. This means facing a weekly scheduling problem, which makes the optimization problem harder as the constraints are harder to satisfy [32].

The particular version of the HCCS recalls the multidimensional knapsack problem [15] with additional constraints. The core of this work has to do with the current demographic trend in many countries, caused by several factors such as low birth rates and increasing life expectation, which has provoked a growing demand of these HC services. From this academic point of view, this means that the large size of instances prevents usage of exact techniques. For this reason this work elaborates on metaheuristics [7], which are stochastic algorithms able to solve complex optimization problems, providing them with accurate solutions in reasonable execution times. Some authors have already used metaheuristics for the HCCS problem, however most of those techniques are basically used to optimize a solution generated by planning techniques [14]. But even metaheuristics take very long running times when dealing with time-consuming fitness evaluations, high number of fitness evaluations, or other reasons. In this context, parallelism is one the the strategies that can be used to reduce the computational times of metaheuristics to affordable values [1]. In this paper, a simple, yet accurate evolutionary algorithm (EA, [2]), concretely, a $(1+\lambda)$ EA has been devised (see [26] for details on this notation). It is a simple form of Evolution Strategy (ES) with no evolution of the parameters of the mutation operator. The main motivation for using this EA is that it works without using any crossover operator, which is a critical issue in highly constrained scheduling problems like HCCS, because the stochastic recombination of two arbitrary plans commonly used within the EA framework might be highly disruptive [21]. Note that this is a particularity of our problem, as the crossover operator may certainly be a desired feature when the problem of destructive recombination is not present [12].

Our goal is to parallelize the $(1+\lambda)$ EA in such a way that it is able to incorporate as many parallel processing elements as possible using a Grid computing system. As several theoretical studies on parallel ES state that $(1+\lambda)$ algorithms have a less-than-linear (logarithmic) speed-up as a function of λ [29,30], many subsequent papers have configured parallel $(1+\lambda)$ EAs with λ large when the parallel computing platform is composed of a large number of processors (e.g., [5]). However, it is also reported in the literature that using large λ values has experimentally shown to perform poorly [6,28]. This is also consistent with our previous work on the HCCS problem [17]. The aim of this work is to engineer and evaluate a parallelization of the $(1+\lambda)$ EA being able to use a large number of distributed processors (to reduce the runtime of the algorithm), but keeping a small value of λ that does not degrade the solution quality. It has to be remarked that this work is intended to be an

extension of [17], in which the algorithm was deployed in a dedicated cluster with up to 32 cores. It improves upon these preliminary experiments with the following novel contributions:

1. The algorithm is now deployed on a Grid computing platform using Condor [31] with dynamic availability of the worker computers. The impact of using 100, 200, and 300 processors in the parallel computation has been measured. A new parallelization strategy has been devised to increase the parallel performance of the algorithm on this new computing facility.
2. A wide set of experiments are also conducted on evaluating different values for λ, also showing that the proposed parallelization is able to profit from a larger parallel computing platform, regardless of the value of this parameter.
3. Both homogeneous and heterogeneous workers using different settings for the mutation operator are devised.
4. Last but not least, the best known solution has been reached for several problem instances, outperforming both that of the previous conference paper [17] and the solution provided by the company.

To the best of our knowledge, items 1 to 3 have not been addressed previously in the literature. Indeed, no parallel $(1 + \lambda)$ EA has been massively parallelized on a Grid computing system with up to 300 workers in parallel. The impact of different λ values on different parallel settings has not been evaluated either. Using heterogeneous workers in such parallel algorithm is also a novelty of this paper, and it aims at allowing the user to apply different settings of the mutation operator in a single execution, thus evaluating different strategies in parallel that may avoid a bad performance of a fixed configuration for a given instance. Indeed, the results have shown that the best homogeneous configuration depends on the instance addressed, but the newly proposed heterogeneous version ranked the second-best in most of the cases.

The paper is structured as follows. The next section formulates the HCCS problem addressed. Section 3 describes both the $(1+\lambda)$ EA and the parallelization proposed. The methodology, instances, and results of the experimentation conducted are included in Section 4. Finally, conclusions are drawn and future lines of research are given in Section 6.

2 The Home Care Crew Scheduling Problem

In this section we describe the formulation of the Home Care Crew Scheduling (HCCS) problem that has been used in the present work. As noted before, in order to evaluate the technique in a real-world scenario, we were provided the actual list of assignments used by the company in their daily operation and the criteria they would like to use to evaluate the solutions.

The same instance of the problem was used in a previous conference paper [17], that was our first attempt at using evolutionary techniques to improve the solution presented by the company. For this reason, this section uses the same notation and mathematical description of the particular type of HCCS problem we are addressing.

In Table 1 we describe the data generated from the company data in terms of four datasets: caregivers ($\mathcal{K}$), customers ($\mathcal{C}$), individual services ($\mathcal{S}$) and groups of services ($\mathcal{G}$). During the preliminary work we used geolocation software to transform the address of all customers into their geographical coordinates, building the set of locations ($\mathcal{L}$) required to estimate travel times.

The most general version of HCCSP includes different types of constraints that can be introduced, such as customer requirements and caregiver skills. Also, services may have synchronization requirements among them [19]. Some authors allow relaxed starting times for services (time windows), while others treat starting times as hard constraints. Many additional variations have been proposed to address the specific requirements posed by real-life scenarios [22]. The particular set of constraints used in this work corresponded to the requirements defined by the company providing the data.

We were particularly interested in an important constraint related to the field of patient preference. We assessed the fact that, in practice, caregiver assignment was not made in a per-service basis; instead, for each customer, the same task was performed by the same caregiver in different days during the week, though not always at the same time of the day. We decided to treat this customer preference as a hard constraint [3]. This was accomplished by introducing the notion of service groups. An assignment was defined as a one-to-one relation $\mathcal{G} \rightarrow \mathcal{L}$, composed by a set of pairings between each service group and the caregiver that performs the services included in that group, that might be performed in different week days. This type of weekly scheduling is harder because the set of infeasible assignments is much larger compared to a problem definition where services are considered individually.

In order to evaluate the quality of a solution, a performance measure must be defined that takes into account the actual financial costs in the application scenario. This measure was then used during the optimization process. Many measures can be factored in the definition of a performance measure for personnel schedul-

Sets	Definition	Indices	Definition
$\mathcal{S} = \{s_i\}$	Set of services	i,j	Index for service
$\mathcal{K} = \{w_k\}$	Set of caregivers	k	Index for caregivers
$\mathcal{D} = \{1 \dots 7\}$	Set of days of the week	d	Index for days
$\mathcal{G} = \{G_0, \dots G_N\}$	Set of service groups	g	Index for groups

Table 1 Key to notation used in this work

ing problems [32]. We used financial measures, in which company policies, labor regulation and problem-specific scenarios must be taken into account. In our case we simplified those factors to three:

- Variable cost of the provided solution based on the length of the schedule of each caregiver, their skill levels and qualifications.
- Marginal costs of the solution, calculated as a fixed amount per caregiver available to the company.
- Costs of the violation of soft constraints, such as the increasing cost of overtime when the caregivers workload exceed the nominal workload of their contracts.

In our scenario, the caregiver set $\mathcal{K}$ was homogeneous in terms of skills and costs, due to the simplicity of the tasks being carried out at the customers' homes. In practice, every caregiver might be assigned to any service group, and the hourly wages and marginal cost per contract were the same for each caregiver.

Regarding variable costs, in the current scenario, caregivers were being paid for the full length of their schedule ("working time") regardless of the actual part of that schedule that was dedicated to actual work ("service time"), traveling from one customer location to the next one ("travel time"), or just time spent waiting for the start of a service ("gaps"). From a certain point of view it seems that the performance measure should make a distinction between both types of "non-productive" time. For instance, if travel time is shorter, a solution may imply a reduction in fuel costs. However, in this case this was not taken into account by the company, so that distinction was not taken into account when calculating the solution performance.

Any realistic solution to the problem must balance variable costs due to the caregiver schedule with the marginal cost of contracting a new caregiver. Variable costs are minimized by decreasing either travel time or gaps. However, the trivial solution of assignation of a different caregiver to each service group is highly impractical due to the marginal costs of hiring such a number of part-time workers. Also, a minimum weekly workload is required by labor regulation, which is one of the constraints we introduced in our formulation.

The performance measure used in the present work aims to the simultaneous minimization of the number of caregivers and the length of the total cost of the week schedules of all the caregivers (see Equations 1 and 2).

2.1 Notation and mathematical description

We present the formal description of our version of HCCSP, following the same notation used in [17]. In Table 2 we list all the parameters required for a specific instance of the problem with its related constraints.

We shall also use x^k_{ijd} as a decision variable that represents assignments; its value is 1 *iif* caregiver k performs service j right after service i; and 0 otherwise. Both services i and j must be performed in the same day d.

The problem objective is to minimize the following two values:

1. f_c,total number of caregivers that perform at least one service.

$$f_c = |\mathcal{K}^*| \text{ , where } \mathcal{K}^* \text{ is defined as:}$$
$$\mathcal{K}^* = \{w_k \in \mathcal{K} \quad , \sum_{\substack{d\in\mathcal{D} \\ \forall i,j\in\mathcal{S}}} x^k_{ijd} \neq 0\} \tag{1}$$

2. f_t, total length of the caregivers daily schedules:

$$f_t = \sum_{d\in\mathcal{D}} \sum_{k\in\mathcal{K}} (\sum_{i\in\mathcal{S}} \sum_{j\in\mathcal{S}} s_{ij} x^k_{ijd} + t_{last^k_d}) \tag{2}$$

subject to the following constraints, each of which is given a cost value based in the degree of noncompliance:

$$x^k_{ijd} \in \{0,1\} \; \forall d \in \mathcal{D}, \forall k \in \mathcal{K}, \forall i,j \in \mathcal{S} \tag{3}$$
$$\sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{S}} x^k_{ijd} = 1 \; \forall d \in \mathcal{D}, \forall i \in \mathcal{S} \tag{4}$$
$$s_{ij} \geq x^k_{ijd}(t_i + d_{ij}) \; \forall d \in \mathcal{D}, \forall k \in \mathcal{K}, \forall i,j \in \mathcal{S} \tag{5}$$
$$min^k_{start} \leq \alpha_i \sum_{j\in\mathcal{S}} x^k_{ijd} \leq max^k_{end} - t_i \quad \forall d \in \mathcal{D}, \forall k \in \mathcal{K}, \forall i \in \mathcal{S} \tag{6}$$
$$min^k_{day} \leq \sum_{i\in\mathcal{S}} \sum_{j\in\mathcal{S}} s_{ij} x^k_{ijd} + t_{last^k_d} \leq max^k_{day} \quad \forall d \in \mathcal{D}, \forall k \in \mathcal{K} \tag{7}$$
$$min^k_{week} \leq \sum_{d\in\mathcal{D}} (\sum_{i\in\mathcal{S}} \sum_{j\in\mathcal{S}} s_{ij} x^k_{ijd}) + t_{last^k_d} \leq max^k_{week} \quad \forall k \in \mathcal{K} \tag{8}$$
$$\sum_{d\in\mathcal{D}} \sum_{i,j\in G_n} x^k_{ijd} \in \{0, |G_n| - 1\} \; \forall G_n \in \mathcal{G}, \forall k \in \mathcal{K} \tag{9}$$

Parameter	Description
α_i	Start time of service i
t_i	time to perform service i
d_{ij}	time to travel from service i to service j (delay)
$s_{ij} = \alpha_j - \alpha_i$	time between the start of service i and the start of service j
$last_d^k$	index of the last service assigned to caregiver w_k on day d
min_{start}^k	earliest start time for services assigned to caregiver w_k due to her contract restrictions
max_{end}^k	latest end time for services assigned to caregiver w_k
min_{day}^k	Minimum working time allocation for any single day for caregiver w_k
max_{day}^k	Maximum working time allocation for any single day for caregiver w_k
min_{week}^k	Minimum working time allocation for the week for caregiver w_k
max_{week}^k	Maximum working time for the week for caregiver w_k

Table 2 Key to parameters for the HCS problem, and guide to the notation used in this work

- Constraint in Eq.3 sets the domains of the decision variables.
- Constraint in Eq.4 guarantees that each service is covered exactly once. That is, there is a single $x_{ijd}^k = 1$ for each value of i, considering the values of all caregivers and days.
- Constraint in Eq.5 checks that the schedule is feasible, that is, a caregiver w_k has enough time to move from service i to service j if $x_{ijd}^k = 1$.
- Constraint in Eq.6 limits the working day of caregivers to be within their working shift, i.e., she has to start working not before the earliest time of her working shift, and start the final service in time to finish it before the end of the working shift.
- Constraint in Eq.7 checks that the total daily schedule length is between the limits for that caregiver.
- Similarly, constraint in Eq.8 checks that the total week schedule for caregivers is between the limits for that caregiver.
- Finally, constraint in Eq.9 ensures that all services in a given group G_n are assigned to the same caregiver, as the number of trips between assignments for all services i in the group will either be 0 or $|G_n| - 1$.

Finally, the fitness function minimized by the EA is, as in the previous work [17], a weighted sum of both objectives plus a penalization term for the violated constraints:

$$f = w_c\ f_c + w_t\ f_t + w_{pen}\ c \tag{10}$$

where f_c and f_t are the objective functions defined above, $w_c = 1$ and $w_t = 10$ are weights that quantify the importance of these functions, c is the overall constraint violation of a solution (aggregating all the values from Eq.3 to 9 into one single value), and $w_{pen} = 1000$ is a penalization factor.

3 Algorithmic approach

In the two following sections, the generic $(\mu + \lambda)$ EA framework is first described and, second, its parallelization. The solution encoding used, the fitness function, and the search operator applied are introduced afterwards.

3.1 $(\mu + \lambda)$ Evolutionary Algorithm

This optimization technique first generates μ initial tentative solutions of the optimization problem at hand. Next, the algorithm perturbs and evaluates these μ solutions at each iteration. This perturbation generates λ new neighbor solutions. Then, the best μ solutions taken from the newly generated λ ones are moved on to the next iteration. An outline of the algorithm is shown in Algorithm 1.

Algorithm 1 Pseudocode of the $(\mu + \lambda)$ EA

```
1: population ← new Population(μ)
2: offspring ← new Population(λ)
3: evaluate(population)
4: while (the stopping condition is not met) do
5:    for i = 1 → λ do
6:       solution ← select(population)
7:       perturbation ← mutate(solution)
8:       evaluate(perturbation)
9:       offspring ← add(perturbation)
10:   end for
11:   population ← bestSolutions(population ∪ offspring)
12: end while
```

As stated before, the configuration used in this work uses a value of $\mu = 1$. The seeding procedure for generating the initial solution and the perturbation operator are the core components defining the exploration capabilities of the $(1 + \lambda)$ EA. The definition of these two procedures is detailed below in Section 3.3.

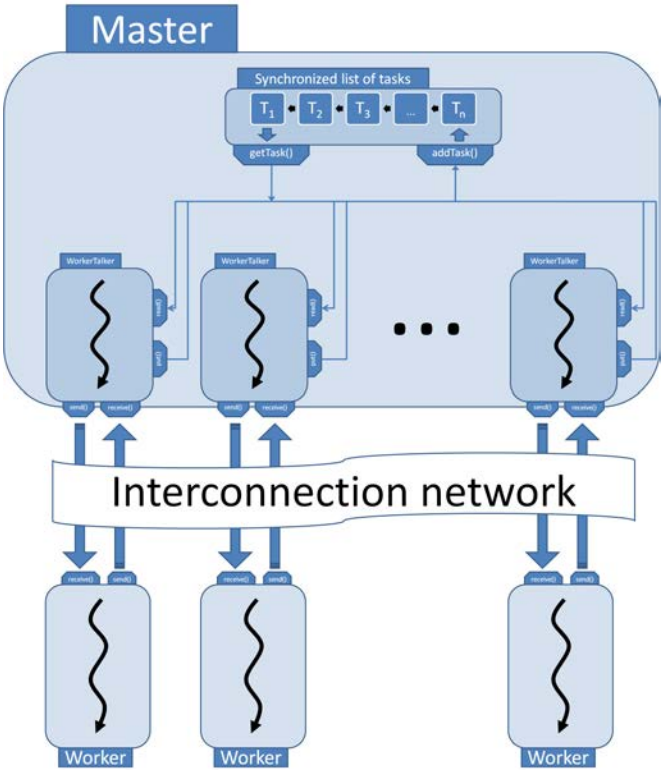

Fig. 1 Underlying parallel software architecture for the $(1+\lambda)$ EA

3.2 Parallel $(1+\lambda)$ EA

A straightforward parallelization of the $(1+\lambda)$ EA would simply be based on sending out the λ neighboring solutions for evaluation to the parallel workers that would eventually return the fitness value of each one. Then, the master gathers all the solutions, picks up the best, and generates λ new solutions which are again passed on to the workers for evaluation. Though simple, this approach has several flaws: first, it can only profit from λ parallel workers at most; second, if the fitness evaluation does not demand high computational requirements, i.e., the ratio computation/communication is not favorable, the master is a bottleneck that reduces the parallel efficiency.

Two main strategies have been developed to address these efficiency issues. The first one aims to relieve the master node from any computation that may be performed in parallel by the workers. In this context, this transfer of computational tasks from the master to the workers is fully achieved by performing the perturbation of the solution remotely on the workers, i.e., the master just transfers the current solution to the workers each of which, upon reception, generates a neighboring solution, computes its fitness, and sends it back to the master (a pseudo-code is shown in Algorithm 2). Note that the newly generated individual must be returned to the master as it might be the new best offspring that should be passed on to the next iteration of the $(1+\lambda)$ EA.

Algorithm 2 Pseudo-code of a worker

```
1: while (not receive finalization notification) do
2:     task ← receiveFromTalker()
3:     solution ← task.solution
4:     newSolution ← perturb(solution)
5:     evaluate(newSolution)
6:     newTask ← new Task(newSolution)
7:     sendToTalker(newTask)
8: end while
```

The second strategy to increase the efficiency of the parallelization is along the line of breaking down the synchronization requirements of the $(1+\lambda)$ EA. The idea is to decouple the number of neighbors generated within the evolutionary loop, i.e., λ, from the number of workers involved in the computation. But before going into the modifications of the $(1+\lambda)$ EA, let us describe the general architecture of the parallel algorithm, which is outlined in Figure 1. As it can be seen, a multi-threaded master has been devised with a talker thread that handles communication with each worker. Talkers and workers communicate via tasks, which are just containers of tentative solutions that are to be evaluated remotely. Tasks are stored in a shared, FIFO list concurrently accessed by all the talkers (in mutual exclusion). Talkers can both remove and add tasks to this list. When the list becomes empty, this means that all the tasks have been processed, i.e., all the solutions have been evaluated by the workers, and the master stops (as well as all the workers).

Algorithm 3 Pseudo-code of the master thread

```
1: currentSolution ← GenerateInitialSolution()
2: offspring ← ∅
3: taskList ← initialize(currentSolution)
4: threads ← runTalkerThreads(currentSolution, offspring,
   taskList)
5: waitForAllThreadsToComplete(threads)
```

The mapping of the $(1+\lambda)$ EA onto this parallel software architecture is as follows. Again, recall that the idea is to take full advantage of a Grid based computing platform composed of many more workers than λ. The master thread allocates all the memory structures of the $(1+\lambda)$ EA, namely the current solution, and the offspring population, and the list of tasks which are remotely computed by the workers (lines 1 to 3 in Algorithm 3). These components are shared (in mutual exclusion) by all the talker threads that are started af-

terwards (line 4). The key point here is that all the EA operations within the evolutionary loop are performed in parallel within the talker threads.

Algorithm 4 Pseudo-code of a talker thread

```
Require: current // the current best solution
Require: offsprings // the solution set for the λ neighbors
Require: taskList // the list with the task to be remotely executed
 1: while (not stopping condition is met) do
 2:    task ← getTask(taskList)
 3:    sendToWorker(task)
 4:    processedTask ← receiveFromWorker()
 5:    neigbor ← processedTaks.solution
 6:    offsprings ← add(neigbor)
 7:    if (offspring.size() == λ) then
 8:       current ← bestSolution(current,offspring)
 9:       offspring ← ∅
10:    end if
11:    if (taskList.size ≤ numWorkers) then
12:       newTask ← new Task(current)
13:       taksList.add(newTaks)
14:    end if
15: end while
```

The two key issues are of the implementation are: on one hand, all the EA components are shared by all the talker threads; on the other hand, all the EA operations within the evolutionary loop are performed in parallel as shown in Algorithm 4. The talker gets (and removes) the first task of the master's task list (line 2), sends it out to its corresponding worker for evaluation (line 3), and waits for the result (line 4). Then, the $(1 + \lambda)$ EA step starts: the newly generated (and evaluated) neighbor is added to the offspring population (in mutual exclusion as many talker threads may be accessing to this shared object) and, if there are as many as λ ones, then the best among the current and the λ neighbors is retrieved (line 8), which becomes the current best one that passes on to the next iteration. Again, it should be remarked that all these operations are performed concurrently by all the talker threads in the master node, so the appropriate mechanism for a reliable access to all the structures has been used. With such an implementation, it can be seen that any number of workers may be involved in the parallel computation, regardless of the value of λ.

Interested readers may find full details directly on the source code, as it is available for downloading at `http://metanet5g.lcc.uma.es/files/mw.zip`. The $(1 + \lambda)$ EA has been implemented using sockets for the remote communications and synchronized statements (see `https://docs.oracle.com/javase/tutorial/essential/concurrency/locksync.html`) to prevent thread interference and memory consistency errors. After some preliminary pilot tests, we have checked that the thread contention overhead is negligible (few seconds) given the settings used in the experimentation performed (up to 300 concurrent threads) and the total running time of the algorithm.

3.3 Solution encoding and search operators

A tentative solution managed by the $(1 + \lambda)$ EA is encoded as an array of integers, p, such that $p[g] = k$ represents, by following the notation of Section 2, the group of services G_g and the caregiver w_k. As a consequence, $g = 1, 2, \ldots, |\mathcal{G}|$ and $k = 1, 2, \ldots, |\mathcal{K}|$. With such a representation, Constraints in in Eq. 3 and in Eq. 9 are directly satisfied.

As stated before, the search capabilities of the $(1 + \lambda)$ EA are mainly given, on the one hand, by the initial solution from which the search starts and, on the other hand, by the perturbation operator used to generate the neighboring solutions. The former issue (seeding) has been addressed by using either the solution already implemented by the company with which this work has been developed. As to the perturbation operator, a neighbors of a solution p is generated by randomly modifying a number of $p[g]$ such that their values are randomly chosen from $\mathcal{G}$, i.e., randomly changing the caregiver assigned to a set of services. The number of services that undergoes modifications is randomly determined by the mutation rate, m_r.

4 Experimentation

This section presents the experimentation performed to evaluate $(1 + \lambda)$ EA when it is deployed on a Grid computing system with up to 300 processing elements. Both the efficacy and the efficiency of the approach have been measured using different values for λ and an increasingly large computing platform, described in the next section. Finally, the obtained results are presented and analyzed.

4.1 Parallel Computing Platform

The $(1 + \lambda)$ EA has been developed in Java, and the underlying grid platform is managed by the Condor system [31]. The technology for implementing the master/worker version is based on standard sockets and Condor is used in turn to gather idle workers and to deploy the workers among the available computing nodes.

The grid computing system used is composed of the computers of the teaching labs of the Department of

Name	Solution size	Caregivers	Quality
WWA	1158	374	2.43e9
WEM	1057	211	4.76e8
WEA	417	121	1.05e9

Table 3 Main features of the HCCS problem instances

Computer Science at the University of Málaga, which are only available from 10:00PM to 8:00AM during the working days, and the entire weekends. Up to 165 machines with an Intel Pentium Dual-Core at 3.2GHz, 8GB of RAM, and Windows 7 (64 bits) have been used. For Condor, each of these cores is a processing slot, so 330 processing elements are indeed available. The interconnection network is a 10GB ethernet. The master node with an Intel Core i5-2320 CPU at 3.00GHz and 16GB of RAM, running Linux 3.2.0-39-generic (x86 64).

4.2 Problem instances and algorithm parameterization

Three different instances of the HCCS problem (Table 3) are used. These instances correspond to three different parts of a weekly schedule: working week and afternoon shift (WWA, 1158 services), weekend and morning shift (WEM 1057 services), and weekend and afternoon shift (WEA, 417 services). The rationale of this division has to do with the terms and conditions of the caregivers' contracts, as they can only work in one of these shifts. Given the fact that there is no overlapping among these instances, they can be considered independent problems.

Table 3 includes their main features: the number of service groups, which determines the size of a solution; the number of caregivers who can serve in the corresponding shift; and, finally, the last column displays the quality of the solution as it is currently implemented by the company, and measured by the fitness function presented in Section 2. This last quality value will be used below as the basis for the comparison of the algorithmic proposal.

As the aim is to improve upon the previous results, the configuration used in the $(1+\lambda)$ EA is based on the best settings evaluated in [17]. From the three values of λ analyzed in that conference paper ($\lambda = 8, 16, 32$), the smaller one, $\lambda = 8$, reached the solutions with the best (lowest) fitness. Starting from this finding, experiments with four different values have been conducted: $\lambda = 1, 2, 4, 8$. The latter value has been kept as a reference to the previous results, and the goal of the three remaining ones is not only to evaluate lower values for λ (looking for better solutions), but also to show that the parallelization proposed is able to profit from the parallel platform, regardless of the number of worker nodes involved in the computation of the $(1 + \lambda)$ EA.

The conclusions drawn from [17] have also driven us to fix two settings of the algorithm. First, the algorithm is seeded with the solution provided by EULEN, rather than using a randomly generated solution. And, second, and more importantly, the workers now do not merely evaluate solutions and send back the fitness. Instead, an hybrid approach is adopted in which the workers perform a straightforward local search based on a $(1 + 1)$ EA. That is, the solution received is repeatedly evaluated and the best between the current and the mutated solution is retrieved. This is performed for a predefined number of steps, which has been set to 1000. This value differs from the one used in [17] (fixed to 10) because a core issue with the HCCS problem is still inherited: solutions have a rather large size (see Table 3), and the time for computing their fitness value is not that long, so, in order to avoid a bottleneck in the master node, the computation/communication ratio has to be increased. It should be remarked that if the computational demands of the underlying optimization problem were large enough, this would not have been required. The point of interest is that even in adverse scenarios, parallelism and Grid computing in general, and the proposed approach in particular, can provide efficient and accurate solutions. Also, even though bottlenecks in the master in master/worker applications are well-known, this paradigm remains quite useful because of its simplicity and efficiency when it is well designed [16].

An actual contribution of this work relies on using heterogeneous workers. Let us elaborate more on this. As the number of workers involved in the parallel computation is, at least, 100, four different mutation rates have been considered, m_r, for the mutation operator, namely, $1/|\mathcal{G}|$ (where $\mathcal{G}$ is number of service groups, i.e., the solution size), 0.01, 0.05, and 0.1. These values have been chosen after some preliminary experiments and taking into account the adopted hybrid strategy. Five different settings have been also defined: four homogeneous, in which all the workers use the same mutation rate of the previous list, and an heterogeneous one in which each mutation rate is used by 25% of the workers. The aim is to show that, without previous knowledge of how a given operator performs for a given problem, using an heterogeneous setting might be helpful because several search strategies can be considered at the same time. Parallelism and Grid computing have clearly enabled us to propose such an approach in this case.

The final parameter to be defined is the stopping condition, which is set as generation of 10000 parallel tasks, i.e., 10000 solutions are sent to the workers, and each one computes 1000 function evaluations. The number of iterations of the $(1 + \lambda)$ EA thus depends on the

λ value. Smaller values of this parameter will allow the algorithm to iterate more times.

It must be acknowledged that the experimental results included below have, however, two known issues that are a direct consequence of the underlying Grid computing system detailed above. On one hand, the intermittent availability of the computing platform (only during nights and weekends) and the wide experimental conditions evaluated (four λ values, five homogeneous plus one heterogeneous search in the workers, three different sizes of the parallel platform, and three instances: $4 \times 5 \times 3 \times 3 = 180$ experiments), have prevented us to provide the results with statistical confidence, as performing the complete experimentation would have taken several years with the current Grid platform. As a consequence, only 5 independent runs of each the $(1+\lambda)$ EA for each setting have been performed. On the other hand, the largest instance addressed in [17] has not been considered because the worker nodes are not able to allocate the Java virtual machine due to memory requirements. Nevertheless, the conclusions are still valid and promising.

5 Results

This section has been structured into two separate subsections, each one devoted to analyzing two different aspects of the parallel $(1 + \lambda)$ EA: the solution quality and the parallel performance.

5.1 Solution quality

Tables 4, 6, and 5 includes the median of the fitness value over 5 independent runs of the hybrid parallel algorithm for the three addressed instances, WWA, WEM, and WEA, respectively. The tables display data for $\lambda = 1, 2, 4, 8$, in three parallel deployments with $w = 100, 200, 300$ workers, and for the five different settings in the worker: one heterogeneous (the "Het" column) and four homogeneous (the columns are named with the mutation rate they use). Let us discuss the most relevant findings that can be drawn from these tables.

Comparing the values of the three tables row by row, i.e., comparing the heterogeneous vs. the homogeneous workers, the configuration that has performed the best (highlighted with a gray background) depends on the considered instance: for the two larger ones, i.e., WWA and WEM, the less disruptive mutation operator with $m_r = 1/|\mathcal{G}|$ has reached the schedules with the smallest (best) fitness values; for the smaller instance, WEA (assigning caregiver to services in the afternoon of the weekends), the second most disruptive mutation with $m_r = 0.01$ (1% of the schedule is randomly modified on average) has reported the highest quality solution. The conclusion is clear, if one knows which operator is the one that best suits for a given problem instance, it will compute the best solutions. However, this information is seldom known in advance and it even changes from instance to instance of a problem. At this point is when heterogeneity comes up as a promising strategy (and a major contribution of this work): in most of the evaluated settings, the configuration of the $(1 + \lambda)$ EA with heterogeneous workers has ranked the second (the exceptions are all for the WWA instance on Table 4, with $\lambda = 1$,$w = 100, 300$; and $\lambda = 2$, $w = 300$), and with tight differences with respect to the best homogeneous configuration. That is: in this case, and with the current experimental setup, this heterogeneous version is recommended if a new instance had to be faced.

			Homogeneous (m_r)			
λ	w	Het	$1/\|\mathcal{G}\|$	0.01	0.05	0.1
1	100	5.162e6	4.230e6	7.091e7	3.156e8	4.423e8
	200	9.830e6	8.559e6	8.728e7	3.147e8	4.292e8
	300	1.389e7	1.236e7	9.581e7	3.206e8	4.356e8
2	100	6.703e6	4.565e6	7.192e7	3.213e8	4.413e8
	200	1.028e7	7.912e6	8.805e7	3.216e8	4.442e8
	300	1.432e7	1.257e7	9.560e7	3.172e8	4.371e8
4	100	6.261e6	4.377e6	7.035e7	3.172e8	4.282e8
	200	1.071e7	8.117e6	8.991e7	3.173e8	4.476e8
	300	1.573e7	1.035e7	9.777e7	3.275e8	4.388e8
8	100	5.964e6	4.221e6	7.080e7	3.166e8	4.322e8
	200	1.125e7	7.768e6	8.726e7	3.111e8	4.357e8
	300	1.357e7	1.224e7	9.756e7	3.173e8	4.472e8

Table 4 Resulting fitness of the $(1 + \lambda)$ EA for the WWA instance

			Homogeneous (m_r)			
λ	w	Het	$1/\|\mathcal{G}\|$	0.01	0.05	0.1
1	100	1.019e8	9.639e7	1.378e8	2.938e8	3.755e8
	200	1.018e8	9.939e7	1.472e8	3.001e8	3.728e8
	300	1.107e8	1.062e8	1.598e8	3.000e8	3.772e8
2	100	9.658e7	9.570e7	1.348e8	2.989e8	3.705e8
	200	1.036e8	1.031e8	1.501e8	2.873e8	3.770e8
	300	1.086e8	1.053e8	1.513e8	3.022e8	3.764e8
4	100	9.715e7	9.194e7	1.342e8	2.881e8	3.762e8
	200	1.041e8	9.500e7	1.458e8	2.996e8	3.732e8
	300	1.106e8	1.031e8	1.562e8	2.973e8	3.750e8
8	100	9.583e7	8.876e7	1.316e8	3.009e8	3.780e8
	200	1.035e8	9.733e7	1.465e8	2.950e8	3.711e8
	300	1.063e8	1.039e8	1.533e8	2.992e8	3.708e8

Table 5 Resulting fitness of the $(1 + \lambda)$ EA for the WEM instance

Concerning the impact of the number of workers, results shown in the three tables are fairly consistent:

λ	w	Het	Homogeneous (m_r)			
			$1/\|\mathcal{G}\|$	0.01	0.05	0.1
1	100	1.234e8	1.232e8	9.975e7	1.629e8	2.314e8
	200	1.186e8	1.251e8	9.777e7	1.586e8	2.260e8
	300	1.279e8	1.268e8	1.010e8	1.672e8	2.320e8
2	100	1.184e8	1.228e8	8.684e7	1.626e8	2.324e8
	200	1.241e8	1.245e8	9.559e7	1.614e8	2.379e8
	300	1.258e8	1.230e8	1.024e8	1.654e8	2.341e8
4	100	1.105e8	1.247e8	9.981e7	1.592e8	2.310e8
	200	1.192e8	1.281e8	9.675e7	1.656e8	2.271e8
	300	1.199e8	1.258e8	9.945e7	1.658e8	2.340e8
8	100	1.136e8	1.195e8	9.373e7	1.626e8	2.230e8
	200	1.213e8	1.248e8	1.044e8	1.679e8	2.249e8
	300	1.241e8	1.367e8	9.735e7	1.640e8	2.315e8

Table 6 Resulting fitness of the (1 + λ) EA for the WEA instance

the larger the number of workers, the worse the quality of the schedule. The reason for such worsening of the solution quality has to do with the induced diversity which is introduced in the parallel model (also essential to the design of a fully parallel algorithm, though). Note that, in order to generate enough individuals for being remotely evaluated and thus reducing the idle time in the workers, whenever a new worker talker receives an evaluated solution, a new task ready for remote execution is created, but based on the current best known solution. As a consequence, as the actual number of workers is much higher than the initially fixed λ value, many of the remote evaluations start with a initial solution that might be worse than any newly incoming solution that is received right after the task is transferred to the worker. This fact clearly holds for the most effective settings, i.e., those with heterogeneous workers and the homogeneous ones with $m_r = 1/|\mathcal{G}|, 0.01$. In the configurations with $m_r = 0.05, 0.1$, the results are rather erratic and no clear conclusion can be drawn. This last issue might be explained by the highly disruptive setting of the mutation operator, that makes the search to become highly stochastic, and int that situation the algorithm finds difficulties to converge. That is, these configurations have a very high mutation rate if the size of the instances considered is taken into account.

With respect to the λ value, and as opposed to the findings reached in [17], quality of schedules reported by the algorithms is not being strongly influenced by setting. Going into the details of the result tables, no clear conclusion can be drawn but, if one does a pure pairwise comparison for each instance between all the settings varying λ and leaving all the remaining ones fixed (e.g., $\lambda = 1, 2, 4, 8$, WWA instance, Het, $w = 100$), this counting reveals that $\lambda = 8$ has performed better than any other setting in 17 out of the 45 comparisons (37.8%), $\lambda = 4$ in 11, $\lambda = 2$ in 8 and, finally, and $\lambda = 1$ have won 8 times. In spite of the tightness of the differences between the different settings, it can be concluded that the setting $\lambda = 8$ has provided the (1 + λ) EA with an accurate balance between exploration and exploitation. Indeed, from the previous results on the problem [17], one might have expected that lower values of λ are better, but the search becomes rather exploitative and gets stuck easily in local optima.

The last part of this section shows the improvements reached with respect to previous results on the problem. First of all, the solutions reached by this Grid-based algorithms represent a better solution (a better final schedule) than those published in [17]. Concretely, for each instance, the best fitness values have been decreased in the following way: for WWA, from 6.60e7 to 4.22e6; for WEM, from 1.73e8 to 8.87e7; and for WEA, from 2.68e8 to 8.68e7. A relevant achievement is reached in the WWA instance (the largest one considered in this work), in which the fitness has been improved by one order of magnitude. It is remarkable, however, that this comparison is not completely fair as here the workers are running a (1+1) EA which uses 100 times more evaluations. In any case, the results are very promising, on the one hand, to the company providing the data (Eulen), as the presented scheduling is much better than the one they have provided (see Table 3) and, on the other hand, to us, as the are still room for further improvements and research in the line of heterogeneity, hybridization, and parallelism on Grid computing.

5.2 Execution times

As in the previous section, results for each instance have been displayed in three separated tables: Tables 7, 8, and 9, which include the median of the execution time of 5 independent runs of the hybrid parallel algorithm for the WWA, WEM, and WEA instances, respectively. First, it is remarkable that, given the features of the parallel computing platform, built upon the Condor system on teaching labs that are not always used in exclusivity, the execution times reported, though meaningful and valid, might have little uncontrolled variations. As much attention as possible has been paid to guarantee identical experimental conditions, but the same remote machines cannot be used for all the executions (as they are automatically allocated by Condor). As these computers are used by students, the software they have installed and their performance may be different, even when the hardware is the same for them all. It should be recalled that, as the parallelization is based on Condor, if for any reason a computer stops its corresponding worker, the parallel task is restarted and the computation of the parallel $(1 + \lambda)$ EA always uses the requested number of workers. This is a well known

			Homogeneous			
λ	w	Het	$1/\|\mathcal{G}\|$	0.01	0.05	0.1
1	100	5450.29	4952.30	5307.72	5428.57	5450.29
	200	2964.16	2745.02	2825.62	2942.32	2964.16
	300	2192.04	2058.90	2150.26	2191.12	2192.04
2	100	5439.84	4959.74	5267.38	5352.92	5439.84
	200	2977.07	2741.38	2822.19	2977.12	2977.07
	300	2188.28	2060.98	2163.44	2184.88	2188.28
4	100	5367.20	4901.23	5241.37	5401.15	5367.20
	200	2962.33	2741.87	2808.83	2925.31	2962.33
	300	2200.34	2057.56	2156.78	2190.67	2200.34
8	100	5401.99	4980.79	5266.06	5395.25	5401.99
	200	2963.86	2749.64	2878.63	2934.27	2963.86
	300	2198.64	2057.66	2142.75	2186.40	2198.64

Table 7 Median running time of the $(1 + \lambda)$ EA for the WWA instance

			Homogeneous			
λ	w	Het	$1/\|\mathcal{G}\|$	0.01	0.05	0.1
1	100	2989.35	2958.96	2991.60	3094.33	3095.06
	200	1722.06	1723.39	1692.28	1709.41	1736.30
	300	1408.76	1375.33	1412.80	1399.83	1408.76
2	100	3009.68	2991.21	3035.80	3088.85	3100.87
	200	1723.48	1709.73	1692.03	1720.24	1757.72
	300	1403.31	1376.28	1391.08	1394.71	1403.31
4	100	2980.16	2938.02	3049.65	3067.93	3089.11
	200	1712.05	1667.71	1722.66	1713.21	1773.76
	300	1402.08	1381.76	1397.21	1416.54	1402.08
8	100	3005.69	2966.62	3042.19	3075.12	3089.71
	200	1721.25	1666.25	1686.21	1710.07	1755.94
	300	1401.77	1380.44	1390.40	1401.09	1401.77

Table 8 Median running time of the $(1 + \lambda)$ EA for the WEM instance

issue when using a Grid computing system and it has been properly considered it so as to minimize its impact on the results.

			Homogeneous			
λ	w	Het	$1/\|\mathcal{G}\|$	0.01	0.05	0.1
1	100	737.82	738.49	754.85	772.35	737.82
	200	646.86	635.95	657.35	647.58	646.86
	300	750.92	750.30	751.05	750.98	750.92
2	100	742.56	741.20	774.75	779.31	742.56
	200	649.38	639.27	644.36	647.40	649.38
	300	753.90	749.80	755.28	763.10	753.90
4	100	786.56	738.13	760.32	744.44	786.56
	200	647.72	637.50	641.20	650.31	647.72
	300	753.68	748.34	752.16	752.10	753.68
8	100	769.91	741.50	761.86	764.07	769.91
	200	650.58	637.86	640.24	659.11	650.58
	300	758.73	754.05	750.32	754.53	758.73

Table 9 Median running time of the $(1 + \lambda)$ EA for the WEA instance

There are several clear conclusions that can be highlighted: when the computation in the workers is expensive enough, the parallelization proposed is able to profit from the full potential of the parallel platform with an increasing number of workers. It is the case of the WWA instance: averaging over all the settings with 100, 200, and 300 workers, the execution times are reduced from 5288.6 down to 2891.3, and to 2158.1, respectively. That is, using twice the number of workers, the reduction has reached 1.8, and when $w = 300$, the average reduction is 2.45. Note that this means roughly, a 90% and 81% of parallel efficiency, a very relevant result taking into account the number of workers involved. Very similar results are achieved in the WEM instance, with 88% and 72% of parallel efficiency. As expected, the smaller instances has reported the worst (lowest) performance with 58% and 33% for $w = 200$ and $w = 300$, respectively. Computation in the workers is not yet expensive enough to reduce the computational times. Indeed, the results with 300 workers for the WWA instance are even longer than those with 100 workers. This is easily explained by the time taken for the algorithm to deploy the workers with Condor. That is, by looking at the traces of the executions, Condor has not been able to deploy the 300 workers when the computation at the master node is already finished. Then, the master has to wait for all the workers to complete, and more workers mean longer waiting times.

Going a little bit more in depth on the results, it can be also stated that the mutation rate does have a clear impact on the execution times. The higher m_r, the more costly the exploration in the workers, and thus the longer the execution times. This is very consistent in all the three instances. It is clear that when $m_r = 1/|\mathcal{G}|$, only one assignment is changed on average, a very fast computational task, and it is the fastest configuration. The heterogeneous setting, as 25% of the workers use each mutation rate, including the most expensive one, it explains the fact that it is the second more slower algorithm.

In general, the λ value has not a clear influence in the execution times. This clearly states a relevant achievement: results show that the parametrization of the search engine in the master node, i.e., the λ value, can be set to the value the user desires. Indeed, there is no need to fix a rather non-usual value in the algorithm to profit from the Grid computing platform (such as using very large population size, many subpopulations, etc.).

As a final remark, and given the tools provided by Condor, additional information about the benefits of the parallelization proposed is shown. Condor reports that, on average, for the WWA instance, the workers have reached an utility value of 97.58%, 95.03%, and 90.00% when using 100, 200, and 300 workers, respectively. The same statistic for the WEM instance is 97.43%, 92.84%, 85.43%, when $w = 100, 200, 300$, respectively. In these two cases, the parallelization de-

vised has been able to keep almost all the workers working during all the computation. Of course, for the WEA instance, these values drop drastically to 87.58%, 65.28%, and 43.48% for each of the three parallel settings. But the truly interesting statistic is the total accumulated computational time by all the workers involved in the execution of the $(1 + \lambda)$ EA. Condor reports that, on average, for the WWA instance, this accumulated time is around 150 hours, i.e., 6.25 days of computation which, in case of using 300 workers, is reduced to 2158 seconds (0.59 hours). This accumulated time falls to around 81 hours for the WEM instance, and to 17 hours for the smaller instance (WEA), being the actual wall-clock times of the execution, for $w = 300$, about 0.48 hours (1400 seconds) and 0.20 hours (750 seconds), respectively.

6 Conclusions and Future Work

This work has addressed very large instances of the Home Care Crew Scheduling problem using a new grid-enabled version of an Evolutionary Algorithm, a $(1 + \lambda)$ Evolutionary Algorithm. Three real-world instances of the problem provided by a leading company in the area have been used. To our knowledge this is the first application of a grid version of such an EA to this type of problem.

While the ES algorithm is a well-known metaheuristic, the proposed approach has been able to introduce two important enhancements over a conventional solution: advanced parallelism and hybridization.

Concerning parallelism, the algorithm has been deployed in up to 300 computing nodes using a master/worker strategy. The parallelization devised is able to profit from such a large computational power without configuring the $(1 + \lambda)$ EA with unusual settings. The resulting algorithm has obtained a high level of parallel efficiency, specially for the most demanding problem (WWE), where it was possible to reach up to 81-90% of parallel performance for the most complex and demanding instance of the problem. This was achieved by designing a distribution strategy that included not only evaluation of solutions, but also performing part of the EA operation in the worker nodes. The strategy is able to distribute optimization tasks as soon as worker nodes are able to accept them, avoiding bottlenecks in the master node. Overall, this means that good results in efficiency do not depend on configuration of the master node.

Concerning results, this work did improve those reported in a previous conference paper, as the current implementation could allocate more computing time for the optimization task. Also, it has been shown that an heterogeneous mix of workers that work with different values of the EA parameters can be a robust solution when there is a lack of apriori knowledge to support parameter selection, which can be a valuable asset for real-world scenarios where it is not allowed to perform extensive preliminary testing for parameters.

This work has also been able to provide insight into some of the mechanisms that are affecting the quality of the solution. Specifically results show that for these instances of the problem, which are already optimized by the providing company, best solutions are obtained when local search is performed gradually by starting from the original solution and using a low value of the mutation rate (m_r) parameter of the EA. This is explained because of the high number of constraints of the problem, where most changes in the solution can degrade its quality by including additional non-compliance with some constraints. Also, results support to some extent that the λ is not as determinant as expected for the final quality of the solution.

As future work, using the efficient parallel approach to develop a more efficient version of the EA will be addressed. The approach can be used to distribute a good solution to working nodes whose local search is heterogeneous not only due to differences in the local search parameters, but in the type of algorithm they run. Also, it is also planned to address the availability and technical issues with the computing platform and memory allocation problems that have prevented us from using the largest of the available instances of the problem.

Acknowledgements The work of Francisco Luna and Juan F. Valenzuela is funded by the Spanish Ministry of Economy, Industry and Competitiveness under contract TIN2016-75097-P. The work of Alejandro Cervantes and Pedro Isasi is funded by the Spanish Ministry of Science and Innovation under contract TIN2011-28336. We also thank EULEN for their contribution and cooperation for this research.

References

1. Alba, E. (ed.): Parallel Metaheuristics. Wiley (2005)
2. Bäck, T.: Evolutionary Algorithms: Theory and Practice. Oxford University Press, New York, USA (1996)
3. Bard, J.F., Shao, Y., Qi, X., Jarrah, A.I.: The traveling therapist scheduling problem. IIE Transactions **46**(7), 683–706 (2014)
4. Bard, J.F., Shao, Y., Wang, H.: Weekly scheduling models for traveling therapists. Socio-Economic Planning Sciences (2013)
5. Belkhir, N., Dréo, J., Savéant, P., Schoenauer, M.: Parameter setting for multicore CMA-ES with large populations. In: Artificial Evolution: 12th International Conference, Evolution Artificielle, pp. 109–122. Springer International Publishing (2016)

6. Beyer, H.G., Sendhoff, B.: Covariance matrix adaptation revisited - The CMSA evolution strategy. In: Parallel Problem Solving from Nature 2008, p. 123?132 (2008)
7. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys **35**(3), 268–308 (2003)
8. Brysy, O., Gendreau, M.: Vehicle routing problem with time windows, part I: Route construction and local search algorithms. Transportation Science **39**(1), 104–118 (2005)
9. Brysy, O., Gendreau, M.: Vehicle routing problem with time windows, part II: Metaheuristics. Transportation Science **39**(1), 119–139 (2005)
10. Carello, G., Lanzarone, E.: A cardinality-constrained robust model for the assignment problem in home care services. European Journal of Operational Research **236**(2), 748–762 (2014)
11. Castillo-Salazar, J., Landa-Silva, D.D., Qu, R.: Workforce scheduling and routing problems: literature survey and computational study. Annals of Operations Research **239**(1), 39–67 (2016)
12. Chen, Z., Wang, R.L.: Ant colony optimization with different crossover schemes for global optimization. Cluster Computing **20**(2), 1247–1257 (2017)
13. Gutiérrez, E., Vidal, C.: Home health care logistics management problems: A critical review of models and methods. Revista Facultad de Ingeniería Universidad de Antioquia **68**, 160–175 (2013)
14. Hiermann, G., Prandtstetter, M., Rendl, A., Puchinger, J., , Raidl, G.: Metaheuristics for solving a multimodal home-healthcare scheduling problem. Central European Journal of Operations Research **23**(1), 89–113 (2015)
15. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer (2004)
16. Linderoth, J., Kulkarni, S., Goux, J.P., Yoder, M.: An enabling framework for master-worker applications on the computational grid. In: Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC), pp. 43 – 50 (2000)
17. Luna, F., Cervantes, A., Isasi, P.: Large-scale home care crew scheduling with a parallel evolutionary algorithm. In: Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2013, pp. 588–593 (2013)
18. Mankowska, D., Meisel, F., Bierwirth, C.: The home health care routing and scheduling problem with interdependent services. Health Care Management Science **17**(1), 15–30 (2014)
19. Masmoudi, M., Mellouli, R.: Milp for synchronized-mtsptw: Application to home healthcare scheduling. In: 2014 International Conference on Control, Decision and Information Technologies (CoDIT), pp. 297–302 (2014)
20. Maya-Duque, P., Castro, M., Srensen, K., Goos, P.: Home care service planning. The case of Landelijke Thuiszorg. European Journal of Operational Research **243**(1), 292 – 301 (2015)
21. Mesman, B.: Genetic algorithms for scheduling purposes. Master's thesis, Eindhoven University of Technology (1995)
22. Rasmussen, M.S., Justesen, T., Dohn, A., Larsen, J.: The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. European Journal of Operational Research **219**, 598 – 610 (2012)
23. Redjem, R., Marcon, E.: Operations management in the home care services: a heuristic for the caregivers' routing problem. Flexible Services and Manufacturing Journal **28**(1), 280–303 (2016)
24. Rendl, A., Prandtstetter, M., Hiermann, G., Puchinger, J., Raidl, R.: Hybrid heuristics for multimodal homecare scheduling. In: CPAIOR 2012, LNCS 7298, pp. 339 – 355 (2012)
25. Rest, K.D., Hirsch, P.: Daily scheduling of home health care services using time-dependent public transport. Flexible Services and Manufacturing Journal **28**(3), 495–525 (2016)
26. Schwefel, H.P.: Numerical Optimization of Computer Models. Wiley (1981)
27. Sooktip, T., , Wattanapongsakorn, N.: Identifying preferred solutions for multi-objective optimization: application to capacitated vehicle routing problem. Cluster Computing **18**(4), 1435–1448 (2015)
28. Teytaud, F.: A new selection ratio for large population sizes. In: EvoApplicatons 2010, vol. 6024 LNCS, pp. 452–460 (2010)
29. Teytaud, F., Teytaud, O.: On the parallel speed-up of estimation of multivariate normal algorithm and evolution strategies. In: pplications of Evolutionary Computing, EvoWorkshops 2009, vol. 5484 LNCS, pp. 655–664 (2009)
30. Teytaud, O., Fournier, H.: Lower Bounds for Evolution Strategies Using VC-Dimension, pp. 102–111. Springer Berlin Heidelberg (2008)
31. Thain, D., Tannenbaum, T., Livny, M.: Condor and the Grid. In: F. Berman, G. Fox, T. Hey (eds.) Grid Computing: Making the Global Infrastructure a Reality. John Wiley & Sons Inc. (2002)
32. Van Den Bergh, J., Belin, J., De Bruecker, P., Demeulemeester, E., De Boeck, L.: Personnel scheduling: A literature review. European Journal of Operational Research **226**(3), 367–385 (2013)
33. Wang, J.Y., Xu, H.C.: Transportation route optimization with cost object in china. Cluster Computing **19**(3), 1489–1501 (2016)
34. Xiang, J., Chen, Z.: An adaptive traffic signal coordination optimization method based on vehicle-to-infrastructure communication. Cluster Computing **19**(3), 1503–1514 (2016)