J Math Model Algor (2007) 6:509–528
DOI 10.1007/s10852-007-9058-5

Parallel Metaheuristics for Workforce Planning

Enrique Alba · Gabriel Luque · Francisco Luna

Received: 1 October 2005 / Accepted: 1 December 2006 /
Published online: 2 March 2007

Abstract Workforce planning is an important activity that enables organizations to determine the workforce needed for continued success. A workforce planning problem is a very complex task requiring modern techniques to be solved adequately. In this work, we describe the development of three parallel metaheuristic methods, a parallel genetic algorithm, a parallel scatter search, and a parallel hybrid genetic algorithm, which can find high-quality solutions to 20 different problem instances. Our experiments show that parallel versions do not only allow to reduce the execution time but they also improve the solution quality.

Key words workforce planning · parallel metaheuristics · parallel genetic algorithm · parallel scatter search · parallel hybrid genetic algorithm

Mathematics Subject Classifications (2000) 68W15 · 90C27 · 90C59

1 Introduction

Decision making associated with workforce planning results in difficult optimization problems because it involves multiple levels of complexity. The workforce planning problem that we tackle in this paper consists of two sets of decisions: selection and assignment. The first step selects an small set of employees from a large number

E. Alba (✉) · G. Luque · F. Luna
Department of Languages and Computational Sciences,
University of Málaga, 29071 Málaga, Spain
e-mail: eat@lcc.uma.es

G. Luque
e-mail: gabriel@lcc.uma.es

F. Luna
e-mail: flv@lcc.uma.es

of available workers and the second (decision) assigns this staff to the tasks to be performed. The objective is to minimize the costs associated to the human resources needed to fulfill the work requirements. An effective workforce plan is an essential tool to identify appropriate workload staffing levels and justify budget allocations so that organizations can meet their objectives.

The complexity of this problem does not allow the utilization of exact methods for instances of realistic size. As a consequence, we firstly propose two parallel metaheuristic methods: a parallel genetic algorithm (GA) and a parallel scatter search (SS). Two kinds of instances have been used to test our approaches. In "structured" ones, there exist a relationship between the tasks duration and the time that a worker can be assigned to them. On the other hand, this constraint is not considered in "unstructured" ones any more, turning these instances more difficult to solve. The development of these methods has the goal of providing a tool for finding high-quality solutions to structured and unstructured instances of the workforce planning problem (WPP). The preliminary results of these two algorithms, where the scatter search approaches outperformed the genetic algorithms [3], have led us to begin to work on the hypothesis that the improvement operator of SS could be the key component provoking these enhancements. Therefore, in this article we present a new hybrid genetic algorithm in which this operator is applied (with certain probability) in its operator pool. The results will confirm our guess and represent a new state of the art tool for this benchmark.

The organization of this paper is as follows. In next section we show a mathematical description of the WPP. In Section 3 and Section 4 we describe the parallel genetic algorithm and the parallel scatter search, respectively. Then, in Section 5 we analyze the results of these algorithms for the solution of the WPP, and finally, we give some hints on future works and conclusions in Section 6.

2 The Workforce Planning Problem

The following description of the problem is taken from Glover et al. [5]. A set of jobs $J = \{1, \ldots, m\}$ must be completed during the next planning period (e.g., a week). Each job j requires d_j hours during the planning period. There is a set $I = \{1, \ldots, n\}$ of available workers. The availability of worker i during the planning period is s_i hours. For reasons of efficiency, a worker must perform a minimum number of hours (h_{min}) of any job to which he/she is assigned and, at the same time, no worker may be assigned to more than j_{max} jobs during the planning period. Workers have different skills, so A_i is the set of jobs that worker i is qualified to perform. No more than t workers may be assigned during the planning period. In other words, at most t workers may be chosen from the set I of n workers and the subset of selected workers must be capable of completing all the jobs. The goal is to find a feasible solution that optimizes a given objective function.

We use the cost c_{ij} of assigning worker i to job j to formulate the optimization problem associated with this workforce planning situation as a mixed-integer program. We refer to this model of the workforce planning problem as WPP:

$$x_{ij} = \begin{cases} 1 \text{ if worker } i \text{ is assigned to job } j \\ 0 \text{ otherwise} \end{cases}$$

$$y_i = \begin{cases} 1 \text{ if worker } i \text{ is selected} \\ 0 \text{ otherwise} \end{cases}$$

$$z_{ij} = \text{number of hours that worker } i \text{ is assigned to perform job } j$$

$$Q_j = \text{set of workers qualified to perform job } j$$

$$\text{Minimize} \sum_{i \in I} \sum_{j \in A_i} c_{ij} \cdot x_{ij} \tag{1}$$

Subject to

$$\sum_{j \in A_i} z_{ij} \leq s_i \cdot y_i \qquad \forall i \in I \tag{2}$$

$$\sum_{i \in Q_j} z_{ij} \geq d_j \qquad \forall j \in J \tag{3}$$

$$\sum_{j \in A_i} x_{ij} \leq j_{max} \cdot y_j \qquad \forall i \in I \tag{4}$$

$$h_{min} \cdot x_{ij} \leq z_{ij} \leq s_i \cdot x_{ij} \qquad \forall i \in I,\ j \in A_i \tag{5}$$

$$\sum_{i \in I} y_i \leq t \tag{6}$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in I,\ j \in A_i$$

$$y_i \in \{0, 1\} \quad \forall i \in I$$

$$z_{ij} \geq 0 \quad \forall i \in I,\ j \in A_i$$

In the model above, the objective function (1) minimizes the total assignment cost. Constraint set (2) limits the number of hours for each selected worker. If the worker is not chosen, then this constraint does not allow any assignment of hours to him/her. Constraint set (3) enforces the job requirements, as specified by the number of hours needed to complete each job during the planning period. Constraint set (4) limits the number of jobs that a chosen worker is allowed to perform. Constraint set (5) enforces that once a worker has been assigned to a given job, he/she must perform such a job for a minimum number of hours. Also, constraint (5) does not allow the assignment of hours to a worker that has not been chosen to perform a given job. Finally, constraint set (6) limits the number of workers chosen during the current planning period.

The same model may be used to optimize a different objective function. Let $\hat{c}_{ij}$ be the cost per hour of worker i when performing job j. Then, the following objective function minimizes the total assignment cost (on hourly basis):

$$\text{Minimize} \sum_{i \in I} \sum_{j \in A_i} \hat{c}_{ij} \cdot z_{ij} \tag{7}$$

Alternatively, p_{ij} may reflect the preference of worker i for job j, and therefore, the following objective function maximizes the total preference of the assignment:

$$\text{Maximize} \sum_{i \in I} \sum_{j \in A_i} p_{ij} \cdot x_{ij} \tag{8}$$

When preference values are used, other objective functions may be formulated. For instance, it may be desirable to maximize the minimum preference value for the set of selected workers. In this paper, we assume that the decision maker wants to minimize the total assignment costs as calculated in Eq. 1.

As pointed out in [10], this problem is related to the capacitated facility location problem (CFLP) as well as the capacitated p-median problem [1, 9]. In fact, our location-allocation problem reduces to a CFLP if the complicating constraints (4), (5) and (6) are relaxed in a Lagrangean manner. In the context of the CFLP, implied bounds are typically added to strengthen the linear programming (LP) relaxation of the mixed-integer programming formulation. The equivalent bounds for the WPP formulation are:

$$x_{ij} - y_i \leq 0 \qquad \forall i \in I,\, j \in A_i \tag{9}$$

Also in the case of the CFLP, an aggregate capacity constraint is usually added to the problem formulation in order to improve some Lagrangean bounds. Even in the case of an LP approach this surrogate constraint can be helpful; it can be used for generating possibly violated lifted cover inequalities. The form of such a constraint for the WPP model is:

$$\sum_{i \in I} s_i \cdot y_i \geq \sum_{j \in J} d_j \tag{10}$$

The difficulty of solving instances of the WPP with an optimization method is related to the relationship between h_{min} and d_j. In particular, problem instances for which d_j is a multiple of h_{min} (referred to as "structured") are easier to handle than those for which d_j and h_{min} are unrelated (referred to as "unstructured").

3 Genetic Algorithm

A genetic algorithm (GA) [7] is an iterative technique that applies stochastic operators on a pool of individuals (the population). Every individual in the population is the encoded version of a tentative solution. Initially, this population is randomly generated. An evaluation function associates a fitness value to every individual indicating its suitability to the problem.

The genetic algorithm that we have developed for the solution of the WPP follows the basic structured shown in Fig. 1. The population size for GAs used in this work is 400 individuals. This value and the specific values of the parameters in the following sections as well have been obtained after preliminary experimentation.

Within the basic structure of the GA for solving the WPP, we have added context information through a special solution representation and crossover operators with improving and repairing mechanisms.

3.1 Representation

Solutions are represented as an $n \times m$ matrix Z, where z_{ij} represents the number of hours that worker i is assigned to job j. In this representation, a worker i is considered

Fig. 1 Basic GA structure

```
Generate (P(0))
t := 0
while not Termination_Criterion(P(t)) do
    Evaluate(P(t))
    P'(t) := Selection(P(t))
    P'(t) := Recombination(P'(t))
    P'(t) := Mutation(P'(t))
    P(t+1) := Replacement(P(t), P'(t))
    t := t+1
return Best_Solution_Found
```

to be assigned to job j if $z_{ij} > 0$. A solution using this representation is shown in Fig. 2. Therefore, the following relationships are established from the values in Z.

$$x_{ij} = \begin{cases} 1 \text{ if } z_{ij} > 0 \\ 0 \text{ otherwise} \end{cases}$$

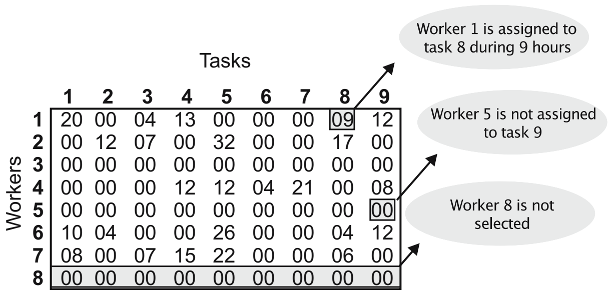
$$y_i = \begin{cases} 1 \text{ if } \sum_{j \in A_i} z_{ij} > 0 \\ 0 \text{ otherwise} \end{cases}$$

3.2 Solution Evaluation

Solutions are evaluated according to the objective function (1) plus a penalty term. The additional term penalizes violations of constraints (2), (3), (4) and (6). The penalty coefficients that are multiplied by the constraint violations are p_2, p_3, p_4, and p_6. Values for these coefficients have been set up to 50, 50, 200, and 800, respectively. Before the fitness value is calculated, new trial solutions are subjected to a repairing/improving operator that makes sure that constraint (5) is satisfied. This operator also ensures that no worker is assigned to a job that he/she is not qualified to perform.

3.3 Repairing/Improving Operator

The purpose of this operator is to repair trial solutions in such a way that they either become feasible with respect to the original problem or at least the infeasibility of these solutions is reduced. The operator performs the four steps outlined in Fig. 3.



Tasks / Workers

	1	2	3	4	5	6	7	8	9
1	20	00	04	13	00	00	00	09	12
2	00	12	07	00	32	00	00	17	00
3	00	00	00	00	00	00	00	00	00
4	00	00	00	12	12	04	21	00	08
5	00	00	00	00	00	00	00	00	00
6	10	04	00	00	26	00	00	04	12
7	08	00	07	15	22	00	00	06	00
8	00	00	00	00	00	00	00	00	00

Worker 1 is assigned to task 8 during 9 hours
Worker 5 is not assigned to task 9
Worker 8 is not selected

Fig. 2 An example of a solution using the representation presented in this work

Fig. 3 Repairing/improving operator

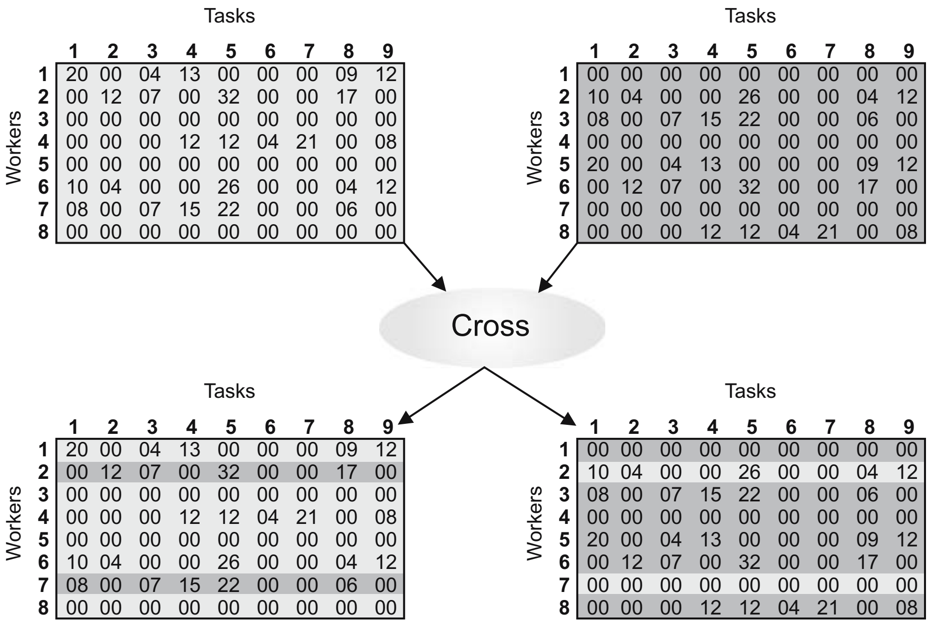
1. Eliminate violations with respect to h_{min}
2. Eliminate violations with respect to assignments of unqualified workers
3. Load feasible workers
4. Reduce infeasibility

In the first step, this operator repairs solutions with respect to the minimum number of hours that a worker must work on any assigned job. The repair is done only on those qualified workers that are not meeting the minimum time requirement. In mathematical terms, if $0 < z_{ij} < h_{min}$ for $i \in I$ and $j \in A_i$, then $z_{ij} = h_{min}$.

The second step takes care of assignments of workers to jobs for which they are not qualified to perform. A value of zero is given to the corresponding entry in Z. Using our notation, if $z_{ij} > 0$ for $i \in I$ and $j \notin A_i$, then $z_{ij} = 0$.

The third step considers that a worker is feasible if he/she satisfies constraints (2) and (4). This step attempts to use up the capacity slack of feasible workers. The slack time for worker i (i.e., $s_i - \sum_{j \in A_i} z_{ij}$) is equally divided among his/her current job assignments. This allows for a higher utilization of the workers that are currently assigned to jobs and thus facilitating the satisfaction of constraint (6).

The last step starts with a partial order of the workers in such a way that those which provoke the largest constraint violation with respect to constraints (2) and (4) tend to appear at the top of the list. This is not a complete order relationship because the operator accounts for a certain amount of randomness in this step. Once the partial order is established, a process of reducing the infeasibility of workers



Workers \ Tasks	1	2	3	4	5	6	7	8	9
1	20	00	04	13	00	00	00	09	12
2	00	12	07	00	32	00	00	17	00
3	00	00	00	00	00	00	00	00	00
4	00	00	00	12	12	04	21	00	08
5	00	00	00	00	00	00	00	00	00
6	10	04	00	00	26	00	00	04	12
7	08	00	07	15	22	00	00	06	00
8	00	00	00	00	00	00	00	00	00

Workers \ Tasks	1	2	3	4	5	6	7	8	9
1	00	00	00	00	00	00	00	00	00
2	10	04	00	00	26	00	00	04	12
3	08	00	07	15	22	00	00	06	00
4	00	00	00	00	00	00	00	00	00
5	20	00	04	13	00	00	00	09	12
6	00	12	07	00	32	00	00	17	00
7	00	00	00	00	00	00	00	00	00
8	00	00	00	12	12	04	21	00	08

Workers \ Tasks	1	2	3	4	5	6	7	8	9
1	20	00	04	13	00	00	00	09	12
2	00	12	07	00	32	00	00	17	00
3	00	00	00	00	00	00	00	00	00
4	00	00	00	12	12	04	21	00	08
5	00	00	00	00	00	00	00	00	00
6	10	04	00	00	26	00	00	04	12
7	08	00	07	15	22	00	00	06	00
8	00	00	00	00	00	00	00	00	00

Workers \ Tasks	1	2	3	4	5	6	7	8	9
1	00	00	00	00	00	00	00	00	00
2	10	04	00	00	26	00	00	04	12
3	08	00	07	15	22	00	00	06	00
4	00	00	00	00	00	00	00	00	00
5	20	00	04	13	00	00	00	09	12
6	00	12	07	00	32	00	00	17	00
7	00	00	00	00	00	00	00	00	00
8	00	00	00	12	12	04	21	00	08

Fig. 4 An example of application of the crossover operator

Fig. 5 Crossover operator

```
for $(i = 1$ to $n)$ do
    if $rand() < \rho_c$ then
        for $(j = 1$ to $m)$ do
            $z^1_{ij} \leftrightarrow z^2_{ij}$
        endfor
    endif
endfor
```

is applied. The process of reducing the violation of constraints (2) and (4) is only applied if it does not provoke new violations of constraints (3) and (5).

3.4 Crossover Operator

A special crossover operator has been designed for the solution of WPP. The operator employs a parameter ρ_c that may be interpreted as the probability that two solutions exchange their current assignments for worker *i*. The process is summarized in Fig. 5 and an example is shown in Fig. 4.

Given two solutions $Z1$ and $Z2$, the crossover operator in Fig. 5 selects, with probability ρ_c, a worker *i*. In the experimentation section, this value is set up to 0.8. If the worker is selected, then the job assignments of solution $Z1$ are exchanged with the assignments of solution $Z2$. The *rand*() function in Fig. 5 generates a uniform random number between 0 and 1.

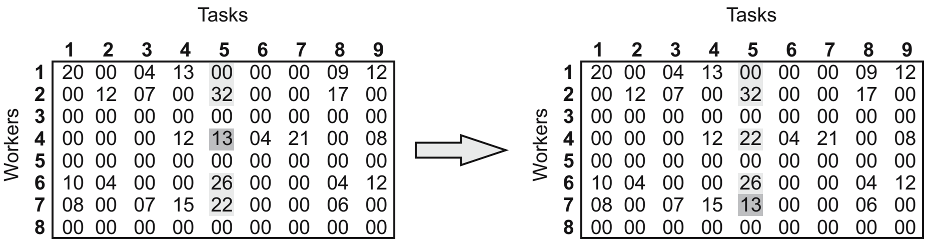
3.5 Mutation Operator

In addition to the crossover operator described above, our GA implementation includes a mutation operator. This mechanism operates on a single solution by exchanging the job assignments of two workers. The job exchange occurs with probability ρ_m, as shown in Fig. 6. An example of this operator is shown in Fig. 7.

Given a solution Z, the mutation operator considers all workers and jobs that the workers are qualified to perform. A random worker *k* is chosen from the list of qualified workers and the exchange of job assignments is considered. For experiments, we set up ρ_m to 0.2. As before, the *rand*() function returns a uniform random number between 0 and 1.

Fig. 6 Mutation operator

```
for $(i = 1$ to $n)$ do
    for $(j \in A_i)$ do
        $k =$ random worker $|k \neq i$ and $k \in Q_j$
        if $rand() < \rho_m$ then
            $z_{ij} \leftrightarrow z_{kj}$
        endif
    endfor
endfor
```



Workers \ Tasks	1	2	3	4	5	6	7	8	9
1	20	00	04	13	00	00	00	09	12
2	00	12	07	00	32	00	00	17	00
3	00	00	00	00	00	00	00	00	00
4	00	00	00	12	13	04	21	00	08
5	00	00	00	00	00	00	00	00	00
6	10	04	00	00	26	00	00	04	12
7	08	00	07	15	22	00	00	06	00
8	00	00	00	00	00	00	00	00	00

Workers \ Tasks	1	2	3	4	5	6	7	8	9
1	20	00	04	13	00	00	00	09	12
2	00	12	07	00	32	00	00	17	00
3	00	00	00	00	00	00	00	00	00
4	00	00	00	12	22	04	21	00	08
5	00	00	00	00	00	00	00	00	00
6	10	04	00	00	26	00	00	04	12
7	08	00	07	15	13	00	00	06	00
8	00	00	00	00	00	00	00	00	00

Fig. 7 An example of application of the mutation operator. The current task assignment for worker 4 and task 5 (*dark-grey background*) is randomly exchanged with the assignment of other qualified worker (worker 7 in this example) of task 5

3.6 Parallel GA

A parallel GA (PGA) [2] is a procedure that consists of multiple copies of an implementation (typically serial) of a genetic algorithm. The individual GAs include an additional communication phase that enables them to exchange information. A PGA is characterized by the nature of the individual GAs and the type of communication that is established among them. Our particular implementation is a distributed GA (dGA), which allows for an efficient exploitation of machine clusters. Typically, dGAs consist of a small number of independent GAs that periodically exchange information. Each individual GA operates on a considerably large population. Since we want to compare against the sequential GA, PGAs use the same population size, but now the whole population of the sequential GA is split into as many subpopulations as processes involved in the parallel computation.

To fully characterize a dGA, the migration policy must be established, which is related to the connection topology of the set of individual GAs. The policy dictates when migration occurs, the number and identity of the individuals that will be exchanged and also determines the synchronization scheme. Our implementation uses a unidirectional ring topology, where each GA receives information from the GA immediately preceding it and sends information to the GA that is immediately after it. At each migration operation which is carried out every 15 generations, one single solution is selected from the population (via binary tournament) and sent to the corresponding neighbor. The newly reached solution replaces the worst individual in the target population if it is better.

4 Scatter Search

Scatter Search (SS) [6] is also a population-based metaheuristic that uses a reference set to combine its solutions and construct others. The method generates a reference set from a population of solutions. Then a subset is selected from this reference set. The selected solutions are combined to get starting solutions to run an improvement procedure. The result of this improvement can motivate the updating of the reference set. A pseudo-code of the scatter search is shown in Fig. 8. The procedures involved by the SS method are the following:

– *Initial population creation:* The first step of this technique is to generate an initial population which is the base set to build the Reference Set. This population

Fig. 8 Basic SS structure

```
generate P
build RefSet from P
while not end do
    generate subsets S from RefSet
    for each subset s in S do
        recombine solution in s to obtain $x_s$
        improve $x_s$
        Update RefSet with $x_s$
    if convergence(RefSet)
        generate a new P
        build Refset from P and the old RefSet
```

must be a wide set of disperse (non similar) solutions. However, it must clearly include also good quality solutions. A simple method to create this population is to use a random generation one (disperse solutions) and then improving some of the solutions to obtain high quality ones. However, other several strategies can be applied to get a population with these properties using problem (heuristic) information.

- *Reference Set update and creation:* SS operates on a small set of solutions, the *RefSet*, consisting of the “good” solutions found during the search. The “good” solutions are not limited to those with the best objective values. By “good” solutions we mean solutions with the best objective values as well as disperse solutions (to escape local optimality and diversify the search). In general, the RefSet is composed of two separate subsets: one subset for the best solutions (RefSet_1) and another for diverse solutions (RefSet_2). This reference set is created from the initial population and it is updated when a new solution is generated. Also, this set is partially reinitialized when the search has stagnated. In our experiments, we use an small RefSet composed of eight solutions ($|RefSet_1| = 5$ and $|RefSet_2| = 3$).
- *Subset generation:* This procedure operates in the reference set to produce subsets of its solutions as a basis for creating combined solutions. In this work, we generate all pairwise combinations of solutions (28 subsets) and then we apply the solution combination operator to them.
- *Solution combination:* It transforms a given subset of solutions into one or more combined solution vectors.
- *Improvement method:* This procedure transforms the current solution into an enhanced solution.

To solve the WPP with SS, we have used the same representation, fitness evaluation, repairing operator, and crossover operator that we used with GA implementation (Sections 3.1, 3.2, 3.3, and 3.4, respectively). These operators have been utilized because they perform an exhaustive and structured search but with new ideas extending SS. The rest of implementation issues are described in the next subsections.

4.1 Initial Population

In our case, the initial population is composed of 15 random solutions which are later enhanced by the improvement method described in the next subsection, and then inserted into the initial population. As in the GA, we want to remark here that the entire parameterization of SS has been tuned properly after preliminary experimentation.

4.2 Improvement Method

A special improvement operator has been designed for the solution of the WPP. The operator employs a parameter ρ_i that may be interpreted as the probability that a worse solution replaces a better solution. The process is summarized in Fig. 9.

Given a solution Z, the improvement operator generates a neighbor (we use the mutation operator described in Section 3.5). If this new solution Z' is better than the original solution Z, we accept that solution and the process is repeated for *MaxIter* iterations. This method also accepts a worse solution with certain probability defined by ρ_i. As before, the *rand*() function returns a uniform random number between 0 and 1, and *fitness*() returns the objective fitness value achieved by a solution. Our SS algorithm will perform 50 iterations of this process and the probability of accepting a worse solution (ρ_i) is 0.1 to escape from local optima.

4.3 Parallel SS

Several parallel implementations of the basic scheme of SS have been proposed in the literature [4]. We are interested in obtaining a parallel method which allows us not only to reduce the execution time but also to improve the solution quality. Hence, we rule out the master-slave model since its numerical performance is the same as the sequential one.

We have used a distributed model, i.e., we have several sequential SS running in parallel that periodically exchange information (one single solution from RefSet). The connection topology is the same as in the PGA. Binary tournament is used for choosing the migrant, which promotes high quality solutions from RefSet$_1$ likely to be selected. In the target SS algorithm, the method for updating the RefSet is applied in order to insert the migrant solution.

In this method, the number of evaluations performed in each step is related to the number of subsets generated. Therefore, we reduce the number of subsets generated by each independent SS so that the computational effort is the same as the sequential version. In concrete, the number of subsets generated is the number of subsets of the serial version (a predefined value) divided by the number of islands. In this case, we choose the subset randomly, but we do not allow the same subset to be selected two or more times.

Fig. 9 Improvement operator

```
**for** $(i = 1 \text{ to } MaxIter)$ **do**
    $Z' = \text{generate neighbor from } Z$
    **if** $fitness(Z') < fitness(Z)$ **or**
       $rand() < \rho_i$ **then**
        $Z = Z'$
    **endif**
**endfor**
```

5 Computational Experiments

In this section we first present the problem instances used. Then, we analyze the behavior of the algorithms with respect to, on the one hand, their ability to find accurate solutions and, on the other hand, the time needed to reach these solutions.

The algorithms in this work have been implemented in C++ and executed on a cluster of Pentium 4 at 2.8 GHz with 512 MB of memory which run SuSE Linux 8.1 (kernel 2.4.19-4GB). The interconnection network is a Fast-Ethernet at 100 Mbps.

5.1 Problem Instances

In order to test the merit of the proposed procedure, we generated artificial problem instances. Given the values of n, m, and t the problem instances were generated with the following characteristics:

$$\begin{aligned}
s_i &= U(50, 70)\\
j_{max} &= U(3, 5)\\
h_{min} &= U(10, 15)\\
\text{Category(worker i)} &= U(0, 2)\\
P(i \in Q_j) &= 0.25 \cdot (1 + \text{Category(worker i)})\\
d_j &= max\left(h_{min}, U\left(\frac{\overline{s}\cdot t}{2\cdot m}, \frac{1.5\cdot\ \overline{s}\cdot t}{m}\right)\right)\\
\text{where } \overline{s} &= \frac{\sum_i s_i}{n} \text{ and } \frac{\sum_j d_j}{\overline{s}\cdot t} \leq \alpha\\
c_{ij} &= |A_i| + d_j + U(10, 20)
\end{aligned}$$

The generator establishes a relationship between the flexibility of a worker and his/her corresponding cost (salary). That is, workers that are able to perform more jobs are more expensive. We solve 10 structured plus other 10 unstructured problems which have been called $s1$ to $s10$ and $u1$ to $u10$, respectively. The ten unstructured problems were generated with the following parameter values: $n = 20$, $m = 20$, $t = 10$ and $\alpha = 0.97$.

Note that the problem generator uses α as the limit for the expected relative load of each worker. The set of ten structured problems was constructed using the same parameter values but h_{min} was set to 4 and the d_j values were adjusted as follows: $d_j = d_j - mod(d_j, 4)$, where $mod(x, y)$ calculates the remainder of x/y. All 20 instances were generated in such a way that a single value for the total number of available hours (s_i) is drawn and assigned to all workers.

5.2 Results: Workforce Planning Performance

The resulting workforce plannings computed by both GA and SS approaches are analyzed in this section. All the algorithms stop after 800,000 function evaluations have been computed. This guarantees that they all are able to converge. The configuration setting is shown in Table 1. These values were obtained experimentally. Values in the tables are average results over 30 independent runs. Since we deal

Table 1 Parameter settings

Parameter	GA	SS
Population size	400	Initial Pop. = 15 RefSet = 8
ρ_c	0.8	–
ρ_m	0.2	–
Subset generated	–	All 2-elements subsets (28)
ρ_i	–	0.1

with stochastic algorithms, we have carried out an statistical analysis of the results which consists of the following steps. First a Kolmogorov–Smirnov test is performed in order to check whether the variables are normal or not. Since all the Kolmogorov–Smirnov normality tests in this work were not successful, we use a non-parametric test: Kruskal–Wallis (with 95% of confidence).

We here remark that the parallel versions of GA and SS have been executed not only in parallel, but also on a single processor. The first reason that motivates these experiments is to check that the parallel search model is independent of the computing platform. As expected, the corresponding tests included in the KW_2 columns of Tables 2 and 3 indicate that no statistical difference exists between them ("–" symbols). In effect, this confirms that they are the same numerical model. As a consequence, in order to compare sequential vs. parallel versions of each algorithm, we have considered only the results of the parallel executions of PGA and PSS (i.e.,

Table 2 GA results for structured and unstructured problems

Prob.	Seq. GA	PGA-4			PGA-8			KW_3
		1 p.	4 p.	KW_2	1 p.	8 p.	KW_2	
*s*1	963	880	879	–	*873*	*873*	–	+
*s*2	994	943	940	–	*920*	922	–	+
*s*3	1,156	*1,013*	1,015	–	1,018	1,016	–	+
*s*4	1,201	1,036	1,029	–	1,008	*1,003*	–	+
*s*5	1,098	1,010	1,012	–	*998*	1,001	–	+
*s*6	1,193	1,068	1,062	–	*1,042*	1,045	–	+
*s*7	1,086	954	961	–	960	*953*	–	+
*s*8	1,287	1,095	1,087	–	*1,068*	1,069	–	+
*s*9	1,107	*951*	956	–	984	979	–	+
*s*10	1,086	932	927	–	*924*	926	–	+
*u*1	1,631	1,386	1,372	–	*1,302*	1,310	–	+
*u*2	1,264	1,132	*1,128*	–	1,153	1,146	–	+
*u*3	1,539	*1,187*	1,193	–	1,254	1,261	–	+
*u*4	1,603	1,341	1,346	–	1,298	*1,286*	–	+
*u*5	1,356	*1,241*	1,252	–	1,254	1,246	–	+
*u*6	1,205	1,207	1,197	–	1,123	*1,116*	–	+
*u*7	1,301	1,176	1,179	–	1,127	*1,121*	–	+
*u*8	*1,106*	1,154	1,151	–	1,123	1,128	–	–
*u*9	1,173	950	938	–	*933*	935	–	+
*u*10	1,214	*1,160*	1,172	–	1,167	1,163	–	+

we do not use the 1p. columns) and therefore the statistical test just involves three datasets (column KW_3). In the second place, running parallel models in a single CPU will allow us to perform the execution time analysis of the algorithms properly (see Section 5.3 for the details). The result of the best algorithm for each instance is marked in "italicized." Let us discuss them in separate sections.

5.2.1 GA Results

The first conclusion that can be drawn from Table 2 is that any PGA configuration is able to solve the considered WPP better than the sequential GA, and statistical confidence exists for this claim (see "+" symbols in column KW_3). The unstructured problem u8 stands for the exception but it can be ruled out since the Kruskal-Wallis test is negative ("−" symbol in column KW_3), thus indicating that the algorithms are not statistically different from each other. Specially accurate solutions have been computed by the PGA in unstructured instances $u1$, $u3$, and $u9$, where the reductions in the planning costs are above 20%.

If we now compare PGAs among them, Table 2 shows that PGA-8 found the best solutions for 13 out of 20 WPP instances, while PGA-4 was only able to find the best plannings in 6 out of 20. This holds specially for the structured problems where PGA-8 gets the best workforce plannings in 8 out of 10 instances. However, it is also noticeable that differences between solutions from PGA-4 and PGA-8 are very small, thus showing that both algorithms have a similar ability for solving the WPP.

Table 3 SS results for structured and unstructured problems

Prob.	Seq. SS	PSS-4			PSS-8			KW_3
		1 p.	4 p.	KW_2	1 p.	8 p.	KW_2	
$s1$	939	896	901	−	*861*	862	−	+
$s2$	952	*904*	905	−	916	913	−	+
$s3$	1,095	1,021	1,019	−	1,005	*1,001*	−	+
$s4$	1,043	1,002	*991*	−	997	994	−	+
$s5$	1,099	*999*	1,007	−	1,009	1,015	−	+
$s6$	1,076	1,031	1,034	−	1,023	*1,022*	−	+
$s7$	987	956	942	−	941	*933*	−	+
$s8$	1,293	1,113	1,120	−	*1,058*	1,062	−	+
$s9$	1,086	*948*	950	−	952	950	−	+
$s10$	945	*886*	891	−	915	909	−	+
$u1$	1,586	1,363	1,357	−	1,286	*1,280*	−	+
$u2$	1,276	1,156	1,158	−	1,083	*1,078*	−	+
$u3$	1,502	1,279	1,283	−	*1,262*	1,267	−	+
$u4$	1,653	1,363	1,356	−	1,307	*1,305*	−	+
$u5$	1,287	1,176	1,192	−	1,175	*1,169*	−	+
$u6$	1,193	1,168	1,162	−	1,141	*1,136*	−	−
$u7$	1,328	1,152	1,151	−	1,084	*1,076*	−	+
$u8$	1,141	1,047	1,039	−	*1,031*	1,033	−	+
$u9$	1,055	906	908	−	886	*883*	−	+
$u10$	1,178	1,003	998	−	*952*	958	−	+

5.2.2 SS Results

We can start analyzing the results of SS (Table 3) in the same way as GA results and state the same conclusions, i.e., parallel SS configurations always get the best solutions versus SS for all the WPP instances and also with statistical confidence ("+" symbols in column KW_3). There are some particular instances in which PSS was able to reduce the planning costs significantly with respect to the sequential SS, e.g. s8, from 1,293 down to 1,048 (reduction of 18%) or u4, from 1,653 down to 1,305 (reduction of 21%). Averaging over structured and unstructured instances, the best PSS configuration reduces WPP costs of sequential SS in 8.35% and 14.98%, respectively.

Turning to compare PSS-4 and PSS-8 between them, Table 3 shows that no conclusion can be draw concerning the structured problems since both algorithms get the best solutions for 5 out of 10 instances. However, PSS-8 always reaches the best workforce planning in the case of the unstructured problems, so (as with PGA-8) we can conclude a slight advantage of PSS-8 over PSS-4.

5.2.3 GA vs. SS

In this section we want to compare both GA and SS approaches for solving WPP. Since there are many different problem instances and analyzing them thoroughly would hinder us from drawing clear conclusions, we have summarized in Table 4 the information of Tables 2 and 3 as follows: we have normalized the resulting planning cost for each problem instance with respect to the worst (maximum) cost obtained by any proposed algorithm, so we can easily compare without scaling problems. Then, values in Table 4 are average values over all the structured and unstructured WPP instances.

A clear conclusion that can be reached is that all SS configurations outperform the corresponding GA ones, that is, considering all structured and unstructured WPP instances, SS gets better solutions than the GA. It is worth mentioning differences between sequential approaches in structured problems (normalized average is reduced from 0.9994 down to 0.9410) and eight island based parallel algorithm in unstructured problems, where PSS-8 normalized costs are 4.4% lower than PGA-8 ones. These results allow us to conclude that SS is a more promising approach for solving this workforce planning problem. Although it can be explained because of the search model of SS by itself, we want to thoroughly discuss this fact. We conjecture that the improvement operator of SS could be responsible for such enhancements since

Table 4 Average results for structured and unstructured problems

Problems		$s1 - s10$		$u1 - u10$	
Algorithm		GA	SS	GA	SS
Sequential		0.9994	0.9410	0.9896	0.9744
	1 p.	0.8858	0.8743	0.8885	0.8605
4 Islands	4 p.	0.8847	0.8747	0.8879	0.8598
	1 p.	0.8783	0.8677	0.8735	0.8308
8 Islands	8 p.	0.8776	0.8663	0.8718	0.8292

adjusting the number of iterations that it performs was the most sensitive parameter in the preliminary experimentation. We will research on this in Section 5.4. Now, let us continue with our analysis, this time from the wall-clock point of view.

5.3 Results: Computational Times

In order to have a fair and meaningful wall-clock time comparison when dealing with such stochastic algorithms, we need to consider exactly the same algorithm and then only change the number of processors, because comparing against the sequential versions would lead to misleading results [2]. Consequently, we have also executed parallel versions of both GA and SS also in a single CPU as shown in Table 5, where we include the average execution times at which the best solution is found during the computation of all the algorithms over 30 independent runs. The same statistical tests have been performed as in the previous section.

If we analyze the execution times of those algorithms being run on a single CPU, it can be seen that sequential optimizers are faster than the monoprocessor execution of any of their parallel version. In order to provide this claim with confidence, we include in column KW$_6$ the result of the statistical test using all the results computed with one single CPU. The "+" symbols in this column indicate that all the execution times are different with statistical significance. This holds for 17 out of 20 instances and 15 out of 20 ones in GA and SS, respectively. The overload of running the several processes of the parallel versions on a single CPU is the main reason for their slower execution. However, sequential algorithms for instances $s6$ and $u8$ in GAs and $s4$, $s7$, $s10$, $u3$, and $u8$ in SS obtain longer execution times than the parallel versions with 4 islands. The point here is that a trade-off exists between the overload due to the number of processes and the ability of the algorithms to easily reach the optimal solution. While the former issue tends to increase the computational time, the latter is a way of reduce it. Results in both tables point out that the computing overload is a very important factor because sequential algorithms usually perform faster than parallel algorithms on one processor.

Analyzing the absolute execution times, one can see the GAs generally get shorter execution times than SS algorithms when the computing platform is composed of just one CPU. However, these differences vanish and even get reversed when we move to actually parallel computing platforms (see columns "4 CPUs" and "8 CPUs" in Table 5). In general, execution times are very similar and differences are not statistically significant in many cases (see "−" symbols in columns KW$_2$).

Two metrics have been used in order to enrich our understanding of the effects of parallelism on the parallel algorithms of this work: the parallel efficiency (η) and the serial fraction (sf) [8]. If we consider that N is the number of processors and s_N is the speedup ($s_N = \bar{t}_{1\ CPU}/\bar{t}_{N\ CPUs}$), the two metrics can be defined as:

$$\eta = \frac{s_N}{N} = \frac{\frac{\bar{t}_{1\ CPU}}{\bar{t}_{N\ CPUs}}}{N} \tag{11}$$

$$sf = \frac{\frac{1}{s_N} - \frac{1}{N}}{1 - \frac{1}{N}} \ . \tag{12}$$

Table 6 includes the resulting values of the metrics. Values of the parallel efficiency show that all the parallel versions of GA and SS are able to profit quite well

Table 5 Execution time (in seconds) for structured and unstructured problems

Pbm	Sequential			4 Islands						8 Islands						KW_6
				1 CPU			4 CPUs			1 CPU			8 CPUs			
	GA	SS	KW_2	PGA-4	PSS-4	KW_2	PGA-4	PSS-4	KW_2	PGA-8	PSS-8	KW_2	PGA-8	PSS-8	KW_2	
*s*1	61	72	+	62	74	+	17	19	+	66	77	+	9	10	+	+
*s*2	32	49	+	32	53	+	9	14	+	37	58	+	6	8	+	+
*s*3	111	114	−	113	118	+	29	31	+	115	127	+	15	17	+	+
*s*4	87	86	−	93	84	+	24	23	−	95	87	+	13	13	−	+
*s*5	40	43	−	41	45	+	13	12	−	46	47	−	9	7	+	+
*s*6	110	121	+	109	122	+	34	33	−	114	128	+	18	18	−	+
*s*7	49	52	+	53	47	+	16	14	+	57	55	−	9	8	+	+
*s*8	42	46	−	45	48	−	13	13	−	48	50	−	7	7	−	+
*s*9	67	70	+	73	71	−	21	19	+	76	74	−	13	10	+	+
*s*10	102	105	+	105	101	+	28	28	−	109	106	+	16	15	−	+
*u*1	95	102	+	98	108	+	29	29	−	102	111	+	16	16	−	+
*u*2	87	94	+	89	95	+	28	26	+	92	99	+	15	14	−	+
*u*3	51	58	+	55	55	−	17	17	−	59	59	−	10	11	+	+
*u*4	79	83	+	79	86	+	26	24	+	86	92	+	15	15	−	+
*u*5	57	62	+	62	62	−	21	18	+	63	68	+	12	10	+	+
*u*6	75	111	+	72	115	+	20	30	+	70	119	+	13	16	+	+
*u*7	79	80	−	81	81	−	24	24	−	89	83	+	15	14	−	+
*u*8	89	123	+	88	118	+	23	35	+	92	123	+	14	20	+	+
*u*9	72	75	−	78	77	−	22	22	−	85	80	+	13	12	−	+
*u*10	95	99	+	96	96	−	25	28	−	99	101	+	13	17	+	+

Table 6 Parallel efficiency and serial fraction for structured and unstructured problems

Problem	PGA-4		PSS-4		PGA-8		PSS-8	
	η	sf	η	sf	η	sf	η	sf
$s1$	0.91	0.032	0.97	0.009	0.91	0.012	0.96	0.005
$s2$	0.88	0.041	0.94	0.018	0.77	0.042	0.90	0.014
$s3$	0.97	0.008	0.95	0.016	0.95	0.006	0.93	0.010
$s4$	0.96	0.010	0.91	0.031	0.91	0.013	0.83	0.027
$s5$	0.78	0.089	0.93	0.022	0.63	0.080	0.83	0.027
$s6$	0.80	0.082	0.92	0.027	0.79	0.037	0.88	0.017
$s7$	0.82	0.069	0.83	0.063	0.79	0.037	0.85	0.023
$s8$	0.86	0.051	0.92	0.027	0.85	0.023	0.89	0.017
$s9$	0.86	0.050	0.93	0.023	0.73	0.052	0.92	0.011
$s10$	0.93	0.022	0.90	0.036	0.85	0.024	0.88	0.018
$u1$	0.84	0.061	0.93	0.024	0.79	0.036	0.86	0.021
$u2$	0.79	0.086	0.91	0.031	0.76	0.043	0.88	0.018
$u3$	0.80	0.078	0.80	0.078	0.73	0.050	0.67	0.070
$u4$	0.75	0.105	0.89	0.038	0.71	0.056	0.76	0.043
$u5$	0.73	0.118	0.86	0.053	0.65	0.074	0.85	0.025
$u6$	0.90	0.037	0.95	0.014	0.67	0.069	0.92	0.010
$u7$	0.84	0.061	0.84	0.061	0.74	0.049	0.74	0.049
$u8$	0.95	0.015	0.84	0.062	0.82	0.031	0.76	0.042
$u9$	0.88	0.042	0.87	0.047	0.81	0.031	0.83	0.028
$u10$	0.96	0.013	0.85	0.055	0.95	0.007	0.74	0.049

from the parallel computing platform. Averaging over all the problems, PGA-4 gets an η value of 0.87, while PSS-4 obtains 0.90. If we consider now the parallelization based on 8 islands, PGA-8 reaches a parallel efficiency of 0.79 whereas PSS-8 achieves a value of 0.85 (also averaging over all the problems). From these average values we can conclude that PSSs profit better from the parallel platform than PGAs although the latter ones are faster in terms of absolute running times.

If we compare the parallel efficiency of the algorithms when the number of processors increases, it can be seen in Table 6 that there is a reduction in the values of this metric and the average values presented previously also support this claim. Here, the serial fraction metric plays an important role. If the values of this metric remain almost constant when using a different number of processors in a parallel algorithm, it allows us to conclude that the loss of efficiency is because of the limited parallelism of the model itself and not because our implementation. For example in the instance $s5$ with PGAs: the parallel efficiency is reduced by 15% (from 0.78 in PGA-4 down to 0.63 in PGA-8) while the serial fraction is almost the same (0.089 in PGA-4 against 0.080 in PGA-8), confirming the previous hypothesis that the loss of efficiency is due to the parallel model.

5.4 A New Hybrid GA

We suggested in Section 5.2.3 that the better workforce planning performance reached by the SS algorithm did lie in the improvement operator used. In order to further investigate this fact, we have developed a new hybrid genetic algorithm (hGA) in which the SS improvement method has been incorporated into the GA main loop as an evolutionary operator. Specifically, the local search algorithm is applied just after the recombination and mutation operators by using a predefined probability which has been set up to $\rho_h = \frac{10}{population\_size}$. The experiments conducted in the following go towards validating the previous claim, so neither the parallel versions are executed on a single processor nor the computational times are presented: only the workforce planning performance is studied for the new hGA, the GA and the SS. This section is therefore intended to be an extension of all the previous work done in the paper. The results of the new hGA are included in Table 7. The stopping condition of all these algorithms is the same as in the previous experiments: reaching a predefined number of function evaluations. In the parallel versions, the hGA also follows the island model described in Section 3.6.

Let us start analyzing the workforce planning performance for the structured instances. As it happened with the GA, the sequential hGA is always outperformed by its two parallel versions. These performance improvements range from 1% in *s*4 to 21.3% in *s*8 (6.46% on average over all the structured instances). Concerning PhGA-4 and PhGA-8, the latter computes the highest performance workforce planning in 6 out of the 10 structured instances, but now differences are smaller (0.75% on average). The diversity introduced by the parallel models is clearly the responsible for this results. The explanation for this claim concerns the loss of diversity provoked by the newly introduced improvement operator and, consequently, the parallel hGA models counteract to some extent the increasingly chance of getting trapped in a local minimum.

In the case of the unstructured instances, the previous claims are even more evident. Here, PhGA-8 gets the best planning performance in 10 out of the 10 instances, and with statistical confidence ("+" symbols in the last column). With respect to

Table 7 Hybrid GA results for structured and unstructured problems

Prob.	Seq. hGA	PhGA-4	PhGA-8	KW_3
*s*1	913	870	*867*	+
*s*2	959	*912*	920	+
*s*3	1,056	1,021	*1,001*	+
*s*4	1,007	999	*997*	+
*s*5	1,103	*998*	1,006	+
*s*6	1,084	1,040	*1,034*	+
*s*7	954	*933*	*933*	+
*s*8	1,295	1,077	*1,067*	+
*s*9	985	*948*	952	+
*s*10	934	903	*891*	+
*u*1	1,375	1,361	*1,280*	+
*u*2	1,193	1,176	*1,098*	+
*u*3	1,509	1,204	*1,187*	+
*u*4	1,670	1,342	*1,286*	+
*u*5	1,189	1,173	*1,170*	+
*u*6	1,193	1,174	*1,128*	+
*u*7	1,288	1,163	*1,106*	+
*u*8	1,076	1,055	*1,041*	+
*u*9	927	894	*883*	+
*u*10	1,205	1,086	*998*	+

Table 8 Average results for structured and unstructured problems

Problems	s1 – s10			u1 – u10		
Algorithm	GA	SS	hGA	GA	SS	hGA
Sequential	0.9989	0.9405	0.9202	0.9735	0.9744	0.9309
4 Islands	0.8842	0.8743	0.8691	0.8877	0.8590	0.8626
8 Islands	0.8771	0.8658	0.8641	0.8711	0.8284	0.8294

the sequential hGA, a noticeable reduction has been reached in $u3$ and $u4$ (27.12% and 29.86%, respectively). Averaging over all the instances, PhGA-8 has been able to reduce the planning costs a percentage of 12.6%. Comparing the two parallel versions, differences now are not that smaller, e.g. reaching almost 9% in $u10$ (from 1,086 down to 998). We can therefore conclude that diversity is even more decisive when solving the unstructured version of the WPP with the new hybrid GA.

In order to compare this new proposal against the previously presented algorithms we have followed the same approach as in Section 5.2.3: we have normalized with respect to all the maximum (worst) values, thus avoiding scaling problems. The results are presented in Table 8. Note that the values for GA and SS are different from those included in Table 4 since the values used for normalization have changed.

If we have a look at the sequential versions of the three algorithms, it can be noticed that hGA outperforms GA and SS in both structured and unstructured instances. This also holds for the parallel models with 4 island in the structured instances. However, the PSS is the best parallel algorithm with 4 islands when solving the unstructured instances (an average value of 0.8590 against 0.8877 of GA and 0.8626 of hGA). The resulting values of the normalized workforce planning performance in the parallel models with 8 islands keep the same behavior: parallel hGA improves upon both parallel GA and parallel SS in the structured instances whereas SS is the best approach in the unstructured ones. The point here is that the differences are tighter. We can conclude that the hGA can profit from using the improvement operator because it always outperforms the GA approach. Concerning SS, hGA is able to always reach improved workforce plannings in the sequential case and in the structured instances.

Summarizing we can conclude that hGA is the best algorithm among all the sequential algorithms used. Concerning parallel versions, a clear conclusion is that parallel versions always outperform serial ones. Among the parallel methods, PGA is the worst while the PSS achieves the best accuracy, obtaining a slightly better performance than PhGA.

6 Conclusions

In this article we have addressed and solved a workforce planning problem. To achieve this goal we have used two parallel metaheuristics: a parallel GA and a parallel SS. The development of these parallel versions of a genetic algorithm and a parallel scatter search aims at tackling problems of realist size.

The conclusions of this work can be summarized attending to different criteria. Firstly, as it was expected, the parallel versions of the methods have reached an important reduction of the execution time with respect to the serial ones. In fact, our parallel implementations have obtained a very good speedup (nearly linear). In several instances, we have noticed a moderate loss of efficiency when increasing the number of processor from four to eight. But this loss of efficiency is mainly due to the limited parallelism of the program, since the variation in the serial fraction was negligible.

Secondly, we have observed that the parallelism did not only allow to reduce the execution time but it also allowed to improve the quality of the solutions. Even when the parallel algorithms were executed in a single processor, they outperformed the serial one, proving clearly that the serial and the parallel methods are different algorithms with different behaviors.

Thirdly, we have noticed that SS results outperformed GA ones for both kind of instances, structured and unstructured ones. The search scheme followed by SS seems to be more appropriate to the WPP than GA one. We have studied whether the improvement operator used by SS is beneficial to this problem, and demonstrated that a hybridization of GA with this local search mechanism provokes an improvement in the quality of the solutions. Indeed, this hybridization has allowed to ameliorate the performance of the "pure" GA in both structured and unstructured instances, whereas it was only able to outperform GA in the structured problems and SS only in the sequential version.

As future work, we plan to apply these techniques to tackle instances ten times larger than those solved here (i.e., $n \approx m \approx 200$).

Acknowledgements The authors are partially supported by the Ministry of Science and Technology and FEDER under contract TIN2005-08818-C04-01 (the OPLINK project).

References

1. Aardal, K.: Capacitated facility location: separation algorithm and computational experience. Math. Program. **81**, 149–175 (1998)
2. Alba, E., Tomassini, M.: Parallelism and evolutionary algorithms. IEEE Trans. Evol. Comput. **6**(5), 443–462 (2002)
3. Alba, E., Luque, G., Luna, F.: Workforce planning with parallel algorithms. IPDPS-NIDISC'06, 246 (2006)
4. García-López, F., Melián-Batista, B., Moreno-Pérez, J., Moreno-Vega, J.: Parallelization of the scatter search. Parallel Comput. **29**, 575–589 (2003)
5. Glover, F., Kochenberger, G., Laguna, M., Wubbena, T.: Selection and assignment of a skilled workforce to meet job requirements in a fixed planning period. In: MAEB'04, pp. 636–641 (2004)
6. Glover, F., Laguna, M., Martí, R.: Fundamentals of scatter search and path relinking. Control Cybern. **39**(3), 653–684 (2000)
7. Holland, J.: Adaptation in Natural and Artificial Systems. (second edition) MIT, Cambridge, Massachusetts (1992)
8. Karp, A., Flatt, H.: Measuring parallel processor performance. Commun. ACM **33**, 539–543 (1990)
9. Klose, A.: An LP-based heuristic for two-stage capacitated facility location problems. J. Oper. Res. Soc. **50**, 157–166 (1999)
10. Laguna, M., Wubbena, T.: Modeling and solving a selection and assignment problem. In: Golden, B., Raghavan, S., Wasil, E. (eds.) The Next Wave in Computing, Optimization, and Decision Technologies, pp. 149–162 (2005)