Workforce Planning with Parallel Algorithms

Enrique Alba, Gabriel Luque, and Francisco Luna

Department of Languages and Computational Sciences
University of Málaga
29071 Málaga, SPAIN
{eat,gabriel,flv}@lcc.uma.es

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 that requires modern techniques to be solved adequately. In this work, we describe the development of two parallel metaheuristic methods, a parallel genetic algorithm and a parallel scatter search, 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.

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 of available workers and the second decision assigns this staff to the tasks that must 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 propose two parallel methods, a parallel genetic algorithm and a parallel scatter search. Two kinds of instances have been used to test our approaches. In "structured" ones, there exists tasks which can be completed without wasting time of employees by definition. On the other hand, this constraint is not considered in "unstructured" ones any more, doing these instances be 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 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.* [3]. 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

1-4244-0054-6/06/$20.00

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} = number of hours that worker i is assigned to perform job j

Q_j = set of workers qualified to perform job j

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

Subject to

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

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

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

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

$$\sum_{i \in I} y_i \leq t \quad (6)$$

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

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

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

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 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) [5] 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 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 to be assigned to job j if $z_{ij} > 0$. 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 the infeasibility of these solutions is reduced. The operator performs the 4 steps outlined in Figure 1.

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

```
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
```

Figure 1. Repairing/improving operator.

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 the most infeasible ones (with respect to constraints (2) and (4)) tend to appear at the top of the list. This is not a complete order because the operator counts on certain amount of randomness for this step. Once the partial order is established, a process of reducing the infeasibility of workers is applied. The process of reducing the violation of constraints (2) and (4) is such that it guards against 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 Figure 2.

Given two solutions $Z1$ and $Z2$, the crossover operator in Figure 2 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 Figure 2 generates a uniform random number between 0 and 1.

```
for (i = 1 to n) do
    if rand() < ρ_c then
        for (j = 1 to m) do
            z^1_ij ↔ z^2_ij
        endfor
    endif
endfor
```

Figure 2. Crossover operator.

```
for (i = 1 to n) do
    for (j ∈ A_i) do
        k = random worker |k ≠ i and k ∈ Q_j
        if rand() < ρ_m then
            z_ij ↔ z_kj
        endif
    endfor
endfor
```

Figure 3. Mutation operator.

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 Figure 3.

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.

3.6 Parallel Version

A parallel GA (PGA) [1] 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 sin-

gle 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) [4] is 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. 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. This population must be a wide set of disperse solutions. However, it must also include good solutions. Several strategies can be applied to get a population with these properties.
- *Reference Set update and creation:* The 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 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 a subset of its solutions as a basis for creating combined solutions. In this work, we generate all 2-elements subsets (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 a trial solution into an enhanced solution.

```
for (i = 1 to MaxIter) do
    Z' = generate neighbor from Z
    if fitness(Z') < fitness(Z) or
       rand() < ρ_i then
        Z = Z'
    endif
endfor
```

Figure 4. Improvement operator.

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 is described in the next subsections.

4.1 Initial Population

In our case, the initial population is composed of 15 random solutions which are enhanced by the improvement method that we describe in the next subsection. 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 in the improvement method. The process is summarized in Figure 4.

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 by mean of a 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 used in the experimentation section performs 50 iterations of this process and the probability of accepting a worse solution (ρ_i) is 0.1.

4.3 Parallel Version

Several parallel implementations of the basic scheme of SS have been proposed in the literature [2]. We are interested in obtaining a parallel method that allows not only to reduce the execution time but also improve

the solution quality. Hence, we rule out the master-slave model.

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, what allows high quality solutions from RefSet_1 more likely to be selected. In the target SS algorithm, the current method for updating the RefSet is applied in order to insert the migrant solution.

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 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.

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:

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

The generator establishes a relationship between the flexibility of a worker and his/her corresponding cost. That is, workers that are able to perform more jobs are more expensive. We solve twenty structured and 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 twenty 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. Values in the tables are average results over 30 independent runs. Since we deal 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. If so, an ANOVA I test is done, otherwise we perform a Kruskal-Wallis test. In fact, all the tests in this work are Kruskal-Wallis tests (with 95% of confidence) since the Kolmogorov-Smirnov normality test did not success any more.

We want to note 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 1 and 2 indicate that no statistical difference exists between them ("−" symbols). 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 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 **boldface**.

5.2.1 GA Results

The first conclusion that can be drawn from Table 1 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

Table 1. 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	
s1	963	880	879	–	**873**	**873**	–	+
s2	994	943	940	–	**920**	922	–	+
s3	1156	**1013**	1015	–	1018	1016	–	+
s4	1201	1036	1029	–	1008	**1003**	–	+
s5	1098	1010	1012	–	**998**	1001	–	+
s6	1193	1068	1062	–	**1042**	1045	–	+
s7	1086	954	961	–	960	**953**	–	+
s8	1287	1095	1087	–	**1068**	1069	–	+
s9	1107	**951**	956	–	984	979	–	+
s10	1086	932	927	–	**924**	926	–	+
u1	1631	1386	1372	–	**1302**	1310	–	+
u2	1264	1132	**1128**	–	1153	1146	–	+
u3	1539	**1187**	1193	–	1254	1261	–	+
u4	1603	1341	1346	–	1298	**1286**	–	+
u5	1356	**1241**	1252	–	1254	1246	–	+
u6	1205	1207	1197	–	1123	**1116**	–	+
u7	1301	1176	1179	–	1127	**1121**	–	+
u8	**1106**	1154	1151	–	1123	1128	–	–
u9	1173	950	938	–	**933**	935	–	+
u10	1214	**1160**	1172	–	1167	1163	–	+

Table 2. 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	1095	1021	1019	–	1005	**1001**	–	+
s4	1043	1002	**991**	–	997	994	–	+
s5	1099	**999**	1007	–	1009	1015	–	+
s6	1076	1031	1034	–	1023	**1022**	–	+
s7	987	956	942	–	941	**933**	–	+
s8	1293	1113	1120	–	**1058**	1062	–	+
s9	1086	**948**	950	–	952	950	–	+
s10	945	**886**	891	–	915	909	–	+
u1	1586	1363	1357	–	1286	**1280**	–	+
u2	1276	1156	1158	–	1083	**1078**	–	+
u3	1502	1279	1283	–	**1262**	1267	–	+
u4	1653	1363	1356	–	1307	**1305**	–	+
u5	1287	1176	1192	–	1175	**1169**	–	+
u6	1193	1168	1162	–	1141	**1136**	–	–
u7	1328	1152	1151	–	1084	**1076**	–	+
u8	1141	1047	1039	–	**1031**	1033	–	+
u9	1055	906	908	–	886	**883**	–	+
u10	1178	1003	998	–	**952**	958	–	+

the Kruskal-Wallis test did not success ("–" symbol in column KW_3), thus indicating that the algorithms are not statistically different from each other. Specially accurate solutions have been computed by PGAs in unstructured instances u1, u3, and u9, where the reductions in the planning costs are greater than 20%.

If we now compare PGAs between them, Table 1 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.

5.2.2 SS Results

We can perfectly start analyzing the results of SS (Table 2) in the same way as GA results, i.e., parallel SS configurations always get the best solutions 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 1293 down to 1048 (reduction of 18%) or u4, from 1653 down to 1305 (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 2 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.

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 3 the information of Tables 1 and 2 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 3 are average values overall 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%

Table 3. 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
4 Islands	1 p.	0.8858	0.8743	0.8885	0.8605
	4 p.	0.8847	0.8747	0.8879	0.8598
8 Islands	1 p.	0.8783	0.8677	0.8735	0.8308
	8 p.	0.8776	0.8663	0.8718	0.8292

Table 4. Execution time (in seconds) for structured and unstructured problems.

	Sequential			4 Islands						8 Islands						KW_6
				1 CPU			4 CPUs			1 CPU			8 CPUs			
Problem	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	
s1	61	72	+	62	74	+	17	19	+	66	77	+	9	10	+	+
s2	32	49	+	32	53	+	9	14	+	37	58	+	6	8	+	+
s3	111	114	−	113	118	+	29	31	+	115	127	+	15	17	+	+
s4	87	86	−	93	84	+	24	23	−	95	87	+	13	13	−	+
s5	40	43	−	41	45	+	13	12	−	46	47	−	9	7	+	+
s6	110	121	+	109	122	+	34	33	−	114	128	+	18	18	−	+
s7	49	52	+	53	47	+	16	14	+	57	55	−	9	8	+	+
s8	42	46	−	45	48	−	13	13	−	48	50	−	7	7	−	+
s9	67	70	+	73	71	−	21	19	+	76	74	−	13	10	+	+
s10	102	105	+	105	101	+	28	28	−	109	106	+	16	15	−	+
u1	95	102	+	98	108	+	29	29	−	102	111	+	16	16	−	+
u2	87	94	+	89	95	+	28	26	+	92	99	+	15	14	−	+
u3	51	58	+	55	55	−	17	17	−	59	59	−	10	11	+	+
u4	79	83	+	79	86	+	26	24	+	86	92	+	15	15	−	+
u5	57	62	+	62	62	−	21	18	+	63	68	+	12	10	+	+
u6	75	111	+	72	115	+	20	30	+	70	119	+	13	16	+	+
u7	79	80	−	81	81	−	24	24	−	89	83	+	15	14	−	+
u8	89	123	+	88	118	+	23	35	+	92	123	+	14	20	+	+
u9	72	75	−	78	77	−	22	22	−	85	80	+	13	12	−	+
u10	95	99	+	96	96	−	25	28	−	99	101	+	13	17	+	+

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 state that the improvement operator of SS could be responsible for such enhancements since adjusting the number of iterations that it performs was the most sensitive parameter in the preliminary experimentation.

5.3 Results: Computational Times

In order to have a fair and meaningful values of these metrics 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 [1]. This way, we have also executed parallel versions of both GA and SS also in a single CPU as shown in Table 4, where we include the average execution times 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 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 this fact. 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 instance u6 in GAs is the extreme case: PGA-8 gets the lowest execution time among the sequential GA and PGA-4, all running on one processor. 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 tries to increase the computational times, the latter is way of reduce them. Results in both tables point out that the computing overload is a more important factor because sequential algorithms usually perform faster.

Table 5. Parallel efficiency and serial fraction for structured and unstructured problems.

	PGA-4		PSS-4		PGA-8		PSS-8	
Problem	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

Analyzing the absolute execution times, one can see the GAs generally get lower 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 4). In general, execution times are very similar and differences are not statistical 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) [6]. 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:

$$\eta = \frac{s_N}{N} = \frac{\frac{\bar{t}_{1\ CPU}}{\bar{t}_{N\ CPUs}}}{N} \quad (7)$$

$$sf = \frac{\frac{1}{s_N} - \frac{1}{N}}{1 - \frac{1}{N}} \ . \quad (8)$$

Table 5 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 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 better profit 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 5 that there is a reduction in the values of this metric and 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. A clear example of this fact is 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) and the serial fraction is almost the same (0.089 in PGA-4 against 0.080 in PGA-8).

6 Conclusions

We have formulated 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 our 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 increase 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.

Finally, 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 think the improvement operator used by SS is beneficial to this problem, and maybe the hybridization of GA with a local search mechanism provokes an improvement in the quality of the solutions. Additional work is required to improve upon the quality of the solutions generated by these methods in the order of ten times larger than those tackled here (i.e., $n \approx m \approx 200$).

Acknowledgments

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

[2] F. García-López, B. Melián-Batista, J. Moreno-Pérez, and J. Moreno-Vega. Parallelization of the Scatter Search. *Parallel Computing*, 29:575–589, 2003.

[3] F. Glover, G. Kochenberger, M. Laguna, and T. Wubbena. Selection and Assignment of a Skilled Workforce to Meet Job Requirements in a Fixed Planning Period. In *MAEB'04*, pages 636–641, 2004.

[4] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39(3):653–684, 2000.

[5] J. Holland. *Adaptation in Natural and Artificial Systems*. The MIT Press, Cambridge, Massachusetts, second edition edition, 1992.

[6] A. Karp and H. Flatt. Measuring Parallel Processor Performance. *Communications of the ACM*, 33:539–543, 1990.