,

HYBRID HEURISTICS FOR THE PERMUTATION FLOW SHOP PROBLEM

M.G. RAVETTI, F.G. NAKAMURA, C.N. MENESES, M.G.C. RESENDE, G.R. MATEUS, AND P.M. PARDALOS

ABSTRACT. The Flow Shop Problem (FSP) is known to be NP-hard when more than three machines are considered. Thus, for non-trivial size problem instances, heuristics are needed to find good orderings. We consider the permutation case of this problem. For this case, denoted by $F|prmu|Cmax$, the sequence of jobs has to remain the same at each machine. We propose and test two hybrid heuristics, combining elements from the standard Greedy Randomized Adaptive Search Procedure (GRASP), Iterated Local Search (ILS), Path Relinking (PR) and Memetic Algorithm (MA). The results obtained are shown to be competitive with existing algorithms.

1. INTRODUCTION

The Flow Shop Problem (FSP) is a scheduling problem in which n jobs have to be processed by m machines. The problem is to find the sequence of jobs for each machine to minimize completion time, also known as makespan (6). This problem is NP-Hard for $m > 3$ (8; 13). Several papers in the literature address this problem, proposing models, heuristics, and bounds. Dannenbring (9) tested several heuristics. Nawaz, Enscore, and Ham (16) presented a polynomial time algorithm (NEH), finding interesting results. Until now, NEH is one of the best polynomial time heuristics for this problem. Taillard (26) presented an improvement in the complexity of the NEH algorithm, a heuristic based on tabu search, and a useful characterization of the distribution of the objective function. Taillard (27) proposed a series of test problems with strong upper bounds. The running time required by Taillard's tabu search heuristic was not given and he focused on solution quality. Ben-Daya and Al-Fawzan (7) implemented and tested an improved variant of Taillard's tabu search, reporting times and comparing their performance with Ogbu and Smith's simulated annealing algorithm (17). However, they did not match all of Taillard's results for large instances. Stützle presented and tested an Iterated Local Search (ILS) heuristic obtaining good results. Ruiz and Maroto (21) compared 25 methods, from very basic ones, such as Johnson's algorithm (12), to more sophisticated ones, such as tabu search and simulated annealing. The results of their study concluded that NEH was the best polynomial-time heuristic, while Stützle's ILS (25) and Reeves's genetic algorithm (18) were the best metaheuristic-based heuristics. Ruiz et al. (22) proposed a new memetic algorithm for this problem, obtaining improved results when compared with the ILS and tabu search. The same authors followed up with another paper in the same direction (23), proposing and testing two genetic algorithms and obtaining strong results. Agarwal, Collak, and Eryarsoy (1) implemented a heuristic improvement procedure based on adaptive learning and applied it to the NEH algorithm, leading to additional improvements. However, for larger instances, their results were of poor quality and their algorithm was computationally intensive. These

Date: November 5, 2006.
Key words and phrases. Heuristics, GRASP, Iterated Local Search, Memetic Algorithm, Flow Shop Problem. AT&T Labs Research Technical Report TD-6V9MEV..

results seemed to present a few issues, that we discuss in Section 5. Ruíz and Stützle (24) described a simple algorithm with which they obtained good results and presented six new upper bounds.

In (1; 7; 27) and almost all other papers dealing with the FSP, the problem of interest was a special case called Permutational Flow Shop Problem (PFSP) in which the jobs have identical ordering sequences on all machines. This widespread approach is useful because it allows a simpler implementation, specially for genetic algorithms, and it is known that a PFSP solution is a good approximation of the FSP solution.

In this paper, we consider the PFSP and propose two hybrid approaches using GRASP-ILS-PR and a memetic algorithm. The paper is organized as follows. In Sections 2 and 3, we describe the basic and the memetic algorithm. In Section 3 we introduce the proposed hybrid algorithms. In Section 5, we present the computational experiments and their analyzes. Finally, in Section 6 we make some concluding remarks.

2. BASIC ALGORITHMS

As it is well known, a traditional GRASP and ILS have two main phases, a construction phase and a local search. Readers unfamiliar with GRASP and ILS are referred to (10; 11; 19) and (14; 25), respectively. In our implementation, we also use path-relinking (20).

2.1. **Construction phase.**

Usually, the construction phase is a greedy algorithm, with some randomness, in which a new solution is obtained at each iteration. In our case, we use the NEH heuristic to construct the first solution. The pseudo-code of the NEH algorithm is presented in Algorithm 1.

Algorithm 1 NEH Algorithm

1: Sort the jobs by decreasing sums of processing times.
2: Schedule the first two jobs minimizing the partial schedule.
3: **for** $i = 3$ **to** n **do**
4: Insert the i-th job in one of the i possible places of the partial solution, minimizing the partial schedule.
5: **end for**

2.2. **GRASP.**

In a standard GRASP, at each iteration a new solution is constructed, usually blending a random procedure with a greedy heuristic. In the heuristic proposed in this paper, we maintain a pool of good-quality solutions on which to apply a path-relinking procedure. After 20 iterations without improvement, we reconstruct the pool using a partial NEH algorithm while preserving most of the structure of the current best solution.

2.3. **ILS.**

As in the implementations of ILS, the GRASP-ILS-PR heuristic has a perturbation phase which changes the current solution by making two-swap movements at random positions. If an improved solution is found, the current solution is updated; if no improvement is found, the current solution is updated with probability $p_T > 0$. This is a simulated-annealing-like acceptance criterion (24; 25). If n is the number of jobs, m is the number of machines, and p_{ij} is the processing time for job i at machine j, then the temperature T used in the acceptance process is

$$T = 0.5 \cdot \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij}}{n \cdot m \cdot 10}.$$

2.4. **Local Search.** We use a well-known local search that has been previously applied to this and other combinatorial optimization problems. Tabu search, ILS, and genetic algorithms have used the same type of local search scheme, e.g. (24). This scheme, which we call `LSInsertion` is based on the insertion procedure of the NEH algorithm. For every job, we remove the job from its original position and attempt to insert it in one of the n possible positions. If an improvement is obtained, then the process is repeated. This procedure works by choosing the positions at random and terminates after a complete search without improvement.

2.5. **Path-relinking.** Path-relinking (PR) is used as a form of intensification of the search procedure. PR is applied from the incumbent solution to a solution randomly selected from the pool (20). Beginning at a random position and considering only swaps, three possible moves are analyzed and the best move is selected. At each step, the new solutions are analyzed and the best one is saved. In Figure 1 an example of the procedure is shown. A similar approach and some variations can be found in (20).

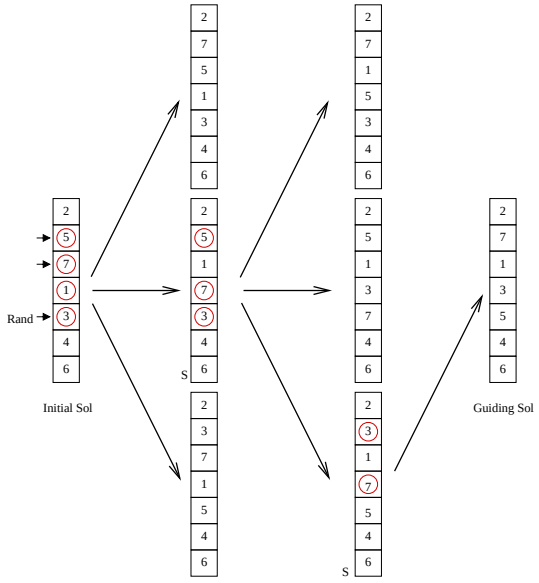

FIGURE 1. Example of the path-relinking procedure. At each step, a random position is chosen and from this point, in a cyclic order, three possible changes are analyzed. The letter 'S' indicates which solution is selected for the next step. This solution corresponds to the best move analyzed. Only five steps are performed if the guiding solution is not reached first.

The PR procedure terminates when the whole path between the solutions is analyzed or after testing a small number of solutions in the path, returning the best solution found in the process. In our implementation, we only allow five steps, and at each step only three possible solutions are analyzed.

The main characteristic of the PR is that at each new step the solutions maintain most of the original structure. When the procedure begins from the best solution, the goal is to preserve most of this successful structure with some influence of the selected pool solution.

2.6. **GRASP-ILS-PR.** Combining these ideas results in the hybrid GRASP-ILS-PR heuristic whose pseudo-code is shown in Algorithm 2. In line 1 of the pseudo-code, the jobs are sorted in decreasing order of the sums of their processing times and the order is saved in the array `JobSorted`. In line 2, the NEH heuristic is called to produce the initial solution that is inserted in the pool in line 3. In line 4, the local search is applied on the initial solution and the local optimum can itself be inserted into the pool in line 5. `NonImproves`, the counter of iterations without incumbent solution improvement, is initialized in line 7.

The loop in lines 8 to 38 constitutes the main iterations of the heuristic. From line 9 to line 11 the current solution `CurrSol` is perturbed and the local search is applied to the perturbed solution. Lines 13 to 21 are only executed every `PRF` iterations, where `PRF` is the path-relinking frequency parameter. In line 13 the path-relinking procedure is applied from the incumbent solution (`BestSolution`) to a solution selected at random from the pool (`PoolSolution`). If certain criteria are met, the path-relinking solution (`SolPR`) is inserted into the pool in line 14. In lines 15 to 21 and 23 to 29 the current solution can be updated when the current solution is improved or with a positive probability otherwise. The pool of solutions is replaced after a certain number of non-improving iterations (`NImp`).

This pool of solutions contains a set of good, or elite, solutions found during the process. As we can see in Algorithm 2, after each local search and after the path-relinking, the algorithm tries to introduce the new solution into the pool. For this insertion, not only the quality of the solution is considered but also the similarity with the other solutions in the pool. This idea is not new and the reader can refer to (3) for example. In our case we do not allow the insertion of solutions that are too similar to the ones already in the pool. A solution is inserted if its objective value is the best so far or if its objective value is better than the worst objective value in the pool and the solution is different enough from the solutions in the pool.

The similarity between two solutions is computed by counting the number of different jobs for a certain position. The importance of a diversified pool is discussed in the next section.

3. MEMETIC ALGORITHM

When working with the PFSP we only deal with a permutation vector, which has an easy representation. This is one of the main advantages of working with the PFSP. The idea of the post-optimization is to use the information gathered by the GRASP-ILS-PR heuristic to search for new solutions near solutions in the pool. For this reason, we use a memetic algorithm (MA) based on the experience of Ruiz and Maroto (21) and Ruiz, Maroto, and Alcaraz (23). The MA works using the pool of solutions produced by the GRASP-ILS-PR as the initial population in addition to a number of random solutions to allow it to search other regions of the feasible solution space.

As the MA is used along with GRASP-ILS-PR, we need a simple structure and only three operators. A mutation procedure and a path crossover, and a cold restart are the only operators used. These operators are described next.

3.1. **Mutation Operator (M).** This operator works as a perturbation approach of GRASP-ILS-PR. Its main goal is to allow the algorithm to search beyond the neighborhood defined

Algorithm 2 GRASP-ILS-PR Algorithm. `Solution`, `CurrSol` and `BestSolution`, represent different solutions and their corresponding makespan values. `PoolSolution` is a random solution chosen from the pool.

```
1: JobSorted ← Sort the jobs by decreasing sums of processing times;
2: Solution ← NEH(JobsSorted);
3: PoolInsertion(Solution);
4: Solution ← LSInsertion(Solution);
5: PoolInsertion(Solution,Proximity);
6: CurrSol ← BestSolution;
7: NonImproves ← 0;
8: for i ← 1,...,MaxIterations do
9:     Solution ← Perturbation(CurrSol);
10:    Solution ← LSInsertion(Solution);
11:    PoolInsertion(Solution,Proximity);
12:    if i mod PRF = 0 then
13:        SolPR ← PathRelinking(BestSolution,PoolSolution);
14:        PoolInsertion(SolPR,Proximity);
15:        if CurrSol > SolPR then
16:            CurrSol ← SolPR;
17:        else
18:            if RND(0,1) < exp(−SolPR − CurrSol)/T then
19:                CurrSol ← SolPR;
20:            end if
21:        end if
22:    end if
23:    if CurrSol > Solution then
24:        CurrSol ← Solution;
25:    else
26:        if RND(0,1) < exp(−Solution − CurrSol)/T then
27:            CurrSol ← Solution;
28:        end if
29:    end if
30:    if BestSolution is improved then
31:        NonImproves ← 0;
32:    else
33:        NonImproves ← NonImproves + 1;
34:    end if
35:    if NonImproves = NImp then
36:        RestartPOOL();
37:    end if
38: end for
```

by the local search. This operator makes two swap moves at random positions. It is applied only to the best or second best solution, randomly chosen in the population.

3.2. **Path Crossover (PX).** This procedure is similar to path-relinking and it is used as a crossover by Ahuja et al. (2) for solving quadratic assignment problems. After the selection of two parents, the crossover consists of the construction of a path between those parents.

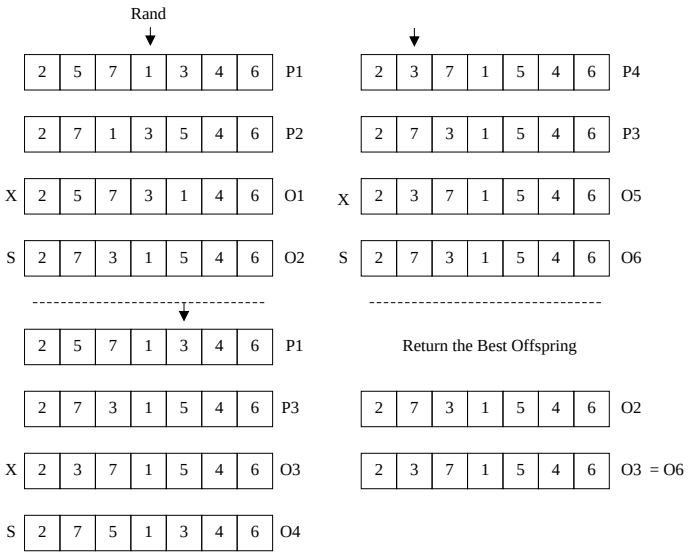

FIGURE 2. PX – *S* indicates the offspring with better objective function and X the discarded offspring. The first position to check is randomly chosen.

This operator begins the procedure from a random position and continues until it reaches this position again. The parent's alleles (jobs) are compared. If they are the same, then the offspring inherits that allele. If the alleles differ, two distinct swap moves can be done, one on each parent. The algorithm always selects the move which results in a better objective function value. The offspring resulting from this crossover is the best solution generated during the construction of the path. Figure 2 shows an example of the PX operator.

We also tested a few other crossover operators, such as multi-parent and similar block order crossovers, but these did not produce results as good as PX or were excessively expensive to compute.

3.3. **Cold Restart.** Cold restart is used in many genetic algorithms with the idea of creating a more diversified population. This is useful when the algorithm reaches a state of stagnation, that is, when no improvement occurs after a certain number of generations. Cold restart changes the population by modifying, replacing, and maintaining part of the solutions. The three best solutions are left unchanged. Approximately half of the population results from applying the mutation operator on one of the three best solutions. About one fourth of the population is generated by applying the iterated greedy algorithm of Ruiz and Stützle (24) to the incumbent solution. The remaining elements of the population are replaced by randomly generated solutions.

4. HYBRID ALGORITHMS

The hybrid algorithms combine the GRASP-ILS-PR and MA heuristics. This combination can be done in several ways, but in this paper we use two approaches called Hybrid A and B.

Initially, the solution of the polynomial-time heuristic NEH is used as the initial pool solution. If this solution can be locally improved, then the improved solution is also inserted into the pool. In both cases the pool of elite solutions is used by the path-relinking component of the hybrid heuristics.

The difference between both hybrid approaches lays in when the MA heuristic is called. In the first algorithm, we use the MA as a post-optimization procedure to intensify the search around the best solutions found during GRASP-ILS-PR phase. After running GRASP-ILS-PR for a fixed number of iterations, the MA uses the pool of elite solutions produced in the GRASP-ILS-PR phase as its initial population. Path-relinking also plays a role in this hybrid heuristic in that it contributes to the pool as well as makes use of elements of the pool for intensification.

In the second approach, the GRASP-ILS-PR algorithm calls the MA algorithm after the algorithm reaches a state of stagnation for a fixed number of iterations. In this case, the MA algorithm uses the pool of elite solutions as the initial population for a certain number of generations. After that, the GRASP-ILS-PR continues to run with the pool provided by the MA output.

The algorithm structures are shown in Figure 3, for now on we will refer to these cases as *Hybrid A* and *Hybrid B*, respectively. The diversity of solutions in the pool is critical in both the GRASP-ILS-PR and MA phases. It allows different promising regions of the solution space to be explored.

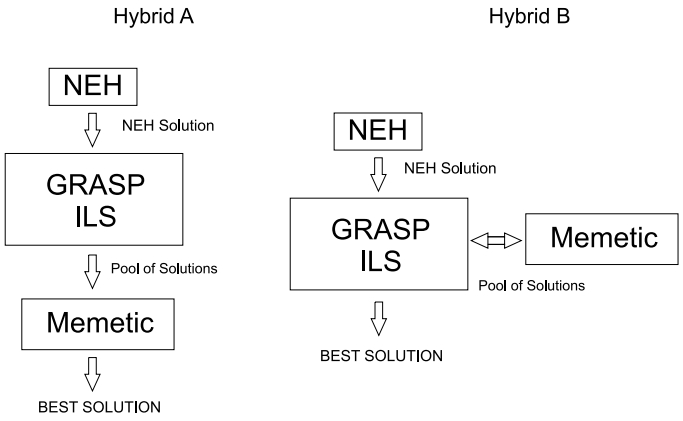

FIGURE 3. Hybrid algorithm structures.

5. COMPUTATIONAL RESULTS

To analyze the performance of the hybrid heuristics, we designed several experiments using benchmark instances proposed by Taillard (26) as our testbed. These experiments are composed of two parts. In the former, we define tests to calibrate certain parameters of our algorithm and in the latter, we analyze the behavior of the algorithm and its performance.

All the algorithms tested in this paper were implemented by the authors in the C programming language and compiled with GNU GCC version 3.2.3, using compiler options `-O6 -funroll-all-loops -fomit-frame-pointer -march=pentium4`. CPU times were

computed using the function `getrusage()`. The experiments for the performance analysis were run on a Dell PowerEdge 2600 computer with dual 3.2 GHz 1 Mb cache XEON III processors and 6 Gb of memory RAM under a Red Hat Linux 3.2.3-53. The algorithm used for random-number generation is an implementation of the Mersenne Twister algorithm described in (15).

5.1. **Calibration.** We first conduct a set of experiments to engineer the algorithm, i.e. set parameters and load balance the various components of the algorithm. Though we conducted extensive computational testing with numerous crossover operators, initial pool solution heuristics, local search algorithms, path-relinking schemes, population and pool sizes, and other parameters, we limit this discussion to the calibration of the memetic algorithm when using PX as a crossover operator, the results of adding path-relinking to the GRASP-ILS-PR heuristic, and the relationship between both components of the hybrid heuristic.

We determine the parameter PRF (frequency of path-relinking), the crossover operator to be used in the MA, and the load balance between the GRASP-ILS-PR and MA components.

5.1.1. *Frequency of path-relinking (PRF).* The GRASP-ILS-PR heuristic works with a pool of 15 solutions. Since path-relinking is not activated at each GRASP-ILS-PR iteration, the main parameter to be analyzed is the frequency in which it is called. This is parameter PRF in Algorithm 2.

To test the effect of using path-relinking, we compared five versions of the hybrid algorithm which differed only with respect to the PRF parameter. This included a variant in which $\mathtt{PRF} = 0$, i.e. the pure ILS proposed by Stützle (25). To perform these tests, we allowed the algorithm to run for 200 CPU seconds, regardless of instance size. The 30 chosen instances have 50, 100, and 200 jobs, with 20 stages, and they proved to be the most difficult of this benchmark. The use of time as a stopping criterion was considered because the path-relinking strategy directly affects CPU time and it is this tradeoff of quality and time that we want to understand.

For each instance tested, each variant was run five times using different random number generator seeds and the average percentage increase over the best known solution (*API*) was calculated. These results are summarized in Table 1. *API* is computed considering the best upper bound known to the date. The formula for *API* is given by

$$API = \frac{1}{R}\sum_{i=1}^{R}\left(\frac{X-UB}{UB}\times 100\right),$$

where R is the number of repetitions, X the solution of the heuristic, and UB the best known upper bound.

The results in Table 1 indicate that GRASP-ILS-PR with path-relinking outperforms the pure ILS. Furthermore, on average the use of path-relinking frequency parameter $\mathtt{PRF} = 2$ results in a better performance than any other frequency tested.

5.1.2. *Crossover comparison.* The memetic algorithm component of the hybrid heuristic has several parameters that need to be set. We conducted extensive experimentation to set these parameters. These experiments resulted in the following parameter choices. The size of the population was set to 20 solutions of which five are randomly generated at each new generation. The probability of applying the crossover is 40%, the mutation 1%, and after applying the operators, local search is carried out with a probability of 5%. To avoid superfluous computations, all the solutions have a flag that indicates if local search

TABLE 1. Average percentage increase over the best known solution for GRASP-ILS-PR varying the path-relinking frequency parameter PRF.

Jobs × Machines	PRF = 2	PRF = 5	PRF = 10	PRF = 20	Pure ILS
	1.36104	1.78701	1.70909	1.49610	1.76623
	2.11123	2.00864	1.77106	1.76026	2.12203
	2.52129	2.39495	2.01044	2.48283	2.42790
	1.57358	1.57358	1.68099	1.83673	1.53598
50×20	1.54528	1.53974	1.47328	1.71144	1.62836
	1.74763	1.81275	1.97558	1.94844	1.78019
	1.57085	1.29015	1.67881	2.04049	1.73819
	2.53048	2.96938	2.86643	2.62260	2.92062
	1.84344	1.68314	1.58696	1.83810	1.46407
	1.29546	1.09371	1.12025	1.22113	1.39634
Average	1.810	1.815	1.787	1.896	1.878
	2.51532	2.56369	2.52499	2.42180	2.73460
	2.17911	2.30197	2.17911	2.17265	2.16618
	2.09217	2.01882	2.03795	2.21336	2.17509
	1.80252	1.82166	1.80252	1.88228	1.80252
100×20	2.30915	2.35350	2.05892	2.22363	1.95439
	2.22816	2.31301	2.20930	2.16216	2.22187
	2.11232	2.23676	2.08679	2.27186	2.34844
	2.78706	2.75269	2.83706	2.56522	2.59334
	2.82709	2.78247	2.76335	2.75697	2.80478
	1.82157	1.82468	1.75008	1.94591	2.04228
Average	2.267	2.297	2.225	2.2616	2.284
	1.74185	1.82939	1.76686	1.82582	1.81510
	2.36722	2.39579	2.37079	2.40650	2.35116
	2.37578	2.54346	2.53993	2.45521	2.45698
	1.98512	2.09846	2.21179	2.23481	2.25961
200×20	1.39977	1.64491	1.72129	1.72484	1.68043
	2.25042	2.22540	2.05917	2.17535	2.17535
	1.92118	1.95109	2.05489	1.88600	1.90887
	2.23222	2.25163	2.28163	2.34869	2.33104
	2.10686	2.20515	2.32666	2.46247	2.34989
	2.56741	2.52851	2.61339	2.77783	2.70179
Average	2.095	2.167	2.195	2.230	2.203
Overall Average	**2.057**	**2.093**	**2.069**	**2.129**	**2.122**
Overall Standard Deviation	0.417	0.444	0.419	0.372	0.411

was applied to the solution. After completing a generation, local search is applied to the best solution with a probability of 10% if its flag indicates that local search has not been previously applied. We next compare the performance of the memetic algorithm when using two crossovers.

After comparing several crossovers, Ruiz, Maroto, and Alcaraz (23) choose the *Similar Block Order Crossover* (SBOX), which presented a better performance than the other operators. The SBOX transfers to the offspring similar blocks of alleles from its parents. Two sequences of alleles are considered a similar block if there are at least two consecutive identical alleles in the same position in both parents. After the recognition of the similar blocks, a crossover point is randomly chosen and the remaining alleles are copied from their parents. By using the crossover point it is possible to generate two offsprings after each crossover. We compare the performance of SBOX with the PX operator described in Section 3.2.

As seen in Table 2, we were unable to reproduce the results obtained by Ruiz, Maroto, and Alcaraz (23). The difference between the performances could be attributed factors such as the stochastic nature of the heuristics, differences in coding, data structure design, compilers, hardware discrepancies, or even computer configuration. Furthermore, Ruiz, Maroto, and Alcaraz report *elapsed* time instead of CPU time.

To evaluate these operators, once again we use the 30 instances from Taillard's benchmark with 50, 100, and 200 jobs and 20 stages. The experiment consisted of allowing each algorithm to perform five runs with different seeds where both algorithms are identical with the exception of the crossover used. Table 2 shows the results, where for each operator we present the *API* and the number of times in which the best solution was found using the crossover.

TABLE 2. Comparison between crossover operators listing *API*, the average percentage increase over the best known solution, and the number of times in which each variant obtained the best solution in the comparison.

	SBOX crossover		PX crossover	
Jobs $\times$ Machines	*API*	times best found	*API*	times best found
50×20	2.28890	4	2.25186	6
100×20	2.60578	6	2.57615	6
200×20	2.38082	3	2.41622	7

We use the 'times best found' metric because sometimes the average value does not show all the information needed to make the correct choice. In this particular case, we can see that for the 200-job instances the variant with the SBOX crossover presents a lower average value but the algorithm using PX crossover obtains better results more times. In these tests, the use of PX as crossover has a better average performance than SBOX, finding best solutions in most cases. All other parameters used in the algorithm were fixed in the comparison.

5.1.3. *Load balancing GRASP-ILS-PR and MA for Hybrid A.* When combining components GRASP-ILS-PR and MA into a hybrid heuristic we need to determine what portion of the total running time will be allocated to each component. We call this choice *load balancing*. We conducted experiments to determine a good load balancing. Once again, we consider the same instances as before and use the 200 CPU seconds as the stopping criterion. The experiment consisted of allowing each variant (with distinct load balance) to perform five runs with different seeds. The possibilities of using the pure GRASP-ILS-PR or the pure MA are also considered in the experiment.

Table 3 and Figure 4 present the results. The figure shows the confidence intervals for the different configurations of the hybrid algorithm. The variants are represented by their respective load balances. For example, variant 95/5 has a load balance with 95% of the CPU time used by the GRASP-ILS-PR and 5% by the MA. As can be seen in the table and figure, the 95%-5% load balance variant has the best average performance.

5.2. **Performance Analysis.** We next investigate the algorithm's dependence on the initial random number generator seed and in its running time. The tables below show the performance of the algorithm after ten runs for each instance. In these experiments, we use number of iterations as the stopping criterion, so that the experiments can be more accurately reproduced.

Hybrid B does not need a load balancing calibration, but after performing similar experimentation two important parameters were set, the algorithm calls the GA only after

TABLE 3. Average relative percentage deviation for different GRASP-ILS-PR and MA load balances. G/M indicates G% of running time allocated to GRASP-ILS-PR and M% of running time allocated to MA.

Jobs × Machines	GRASP-ILS-PR	95/5	80/20	50/50	20/80	5/95	Memetic
	1.39221	1.36104	1.53247	1.48052	1.75065	1.71429	1.91169
	2.20302	2.11123	2.23542	1.87905	2.32721	2.23002	2.12203
	2.58720	2.52129	2.52678	2.48833	2.55974	2.60368	2.50481
	1.76155	1.57358	1.79914	1.69710	1.79914	2.05693	2.10526
50×20	1.78898	1.54528	1.78898	1.78344	1.91083	2.07699	2.06037
	2.05156	1.74763	2.05156	2.05156	2.05156	2.10583	2.09498
	1.76518	1.57085	1.79757	1.74359	1.84615	1.81916	2.35358
	2.56841	2.53048	2.56299	2.56299	2.91520	3.00732	3.07234
	1.85413	1.84344	1.85413	1.85413	1.91825	1.82207	2.23350
	1.46005	1.29546	1.39634	1.41757	1.40165	1.08840	2.05999
Average	1.94323	1.81003	1.95454	1.89583	2.04804	2.05247	2.25186
	2.47662	2.51532	2.75395	2.72170	2.73460	2.88617	3.02806
	2.16618	2.17911	2.26964	2.54122	2.64791	2.75461	3.01326
	2.09217	2.09217	2.14958	2.21017	2.16233	2.58970	2.49721
	1.81528	1.80252	1.80890	1.83442	1.73233	2.04498	1.88547
100×20	2.27431	2.30915	2.36300	2.50871	2.41368	2.22997	2.44219
	2.23130	2.22816	2.22816	2.32558	2.70585	2.69956	2.66813
	2.11232	2.11232	2.09317	2.17294	2.29419	2.53350	2.59413
	2.85268	2.78706	2.85893	2.99328	3.06827	3.16825	2.74957
	2.82709	2.82709	2.83665	2.89721	2.87809	3.04064	3.05976
	1.82157	1.82157	1.89307	2.02362	2.15107	1.83712	2.11999
Average	2.26695	2.26745	2.32551	2.42289	2.47883	2.57845	2.60578
	1.81331	1.74185	1.81331	1.82760	1.91157	1.83117	1.95266
	2.50647	2.36722	2.49755	2.55467	2.46006	2.41899	2.53861
	2.52405	2.37578	2.53993	2.54699	2.52758	2.46404	2.60524
	2.27554	1.98512	2.35346	2.52701	2.44200	2.43492	2.62617
200×20	1.57030	1.39977	1.66800	1.65024	1.78169	1.64491	1.89182
	2.37912	2.25042	2.38806	2.39878	2.29154	2.32192	2.47386
	2.06193	1.92118	2.08128	2.03730	2.00739	1.79803	2.17628
	2.25693	2.23222	2.24634	2.38574	2.45280	2.54632	2.54632
	2.18728	2.10686	2.28020	2.20872	2.34811	2.44282	2.41780
	2.68411	2.56741	2.73362	2.78490	2.86270	2.70179	2.93343
Average	2.22590	2.09479	2.26018	2.29220	2.30854	2.26049	2.41622
Overall Average	**2.14536**	**2.05742**	**2.18007**	**2.20364**	**2.27847**	**2.29714**	**2.42462**

a stagnation of 20 iterations, and it is allowed to run 20 generations after returning the improved pool of elite solutions.

5.2.1. *Time to target.* Many local search based combinatorial optimization heuristics, including GRASP, GRASP with path-relinking, and memetic algorithms have running time to the optimal solution that are distributed according to a shifted exponential distribution (4). To study the random variable *time to target solution value* we ran Hybrid A and Hybrid B using one hundred different seeds for one instance with 100 jobs and 20 stages[1]. The seeds were randomly generated and are distinct from the ones used in the calibration process.

Hybrid A uses 95% of the processing time allowed with GRASP-ILS-PR and the last 5% with the MA. In this test the maximum CPU time allowed to each run is 10,000 seconds. For the Hybrid B algorithm, the MA is called after a period of 20 iterations without improving the incumbent solution, and after 20 generations the MA algorithm returns the pool to the GRASP-ILS-PR algorithm.

A target value of 2.5% above the best known solution was chosen for this experiment. For this particular case, the best known solution has a value of 6314, therefor the target

[1]The instance used is this experiment is one from Taillard's benchmark, called `ta085`.

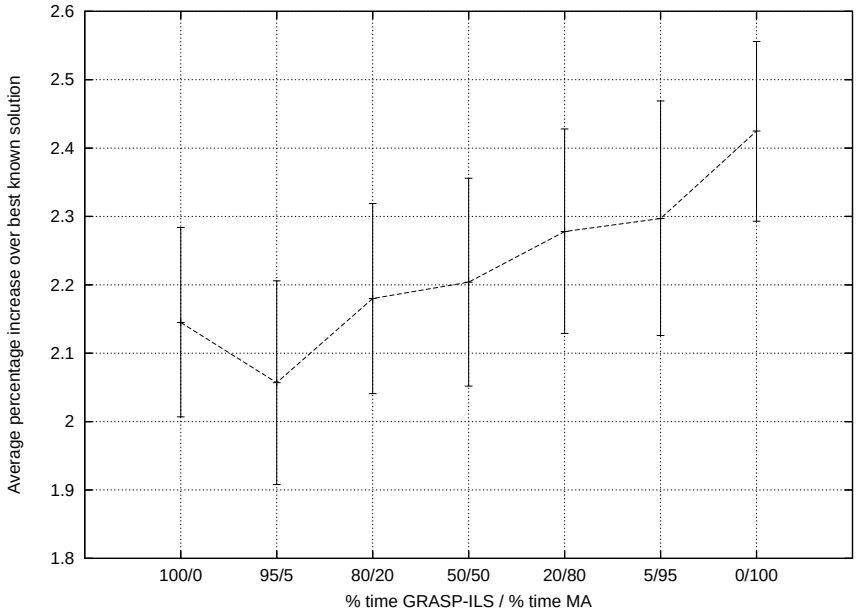

FIGURE 4. 95% confidence intervals for hybrid algorithms.

was set to 6471. The algorithms stop after obtaining a solution less or equal to 6471 or after 10,000 seconds. Time-to-target plots (ttt-plots) are produced for these values using (5). Figure 5 shows the ttt-plots obtained.

Of interest is that within 80 seconds, almost 80% of the cases for both algorithms reached the target and within 1000 seconds more than 90%. These results show that the choice of the seed is not a critical aspect for obtaining a reasonable performance. By this comparison it is possible to conclude Hybrid A presents a performance slightly better than Hybrid B.

5.2.2. *Performance.* In this section, we present the results obtained by using our algorithm in the 120 instances from Taillard's Benchmark. The algorithm performs a fixed number of iterations for both the GRASP-ILS-PR and the memetic algorithm components.

Some authors use a relation between the number of jobs and the number of machines to set a time for testing the performance of the algorithms. The problem with this approach is that for this particular benchmark, the difficulty of the instances does not seem to increase with an increase in the number of jobs. We resume this discussion in next section. In Tables 4 and 5, the average values of our experiments are presented, for both hybrid algorithms. Not only are these results competitive, but it is possible to see that the standard deviation (StdDev) is low, indicating a robust method. When comparing the hybrid approaches, it is possible to observe, the performance of Hybrid A is slightly better than Hybrid B, which is consistent with the time to target experiment.

5.2.3. *Remarks regarding the instances.* It is interesting to analyze the difficulty of our algorithms for some instances. It is quite obvious and expected that the increase in the

FIGURE 5. CPU time (seconds) needed to reach a solution no further than 2.5% of the best known upper bound. The time is on a logarithmic scale.

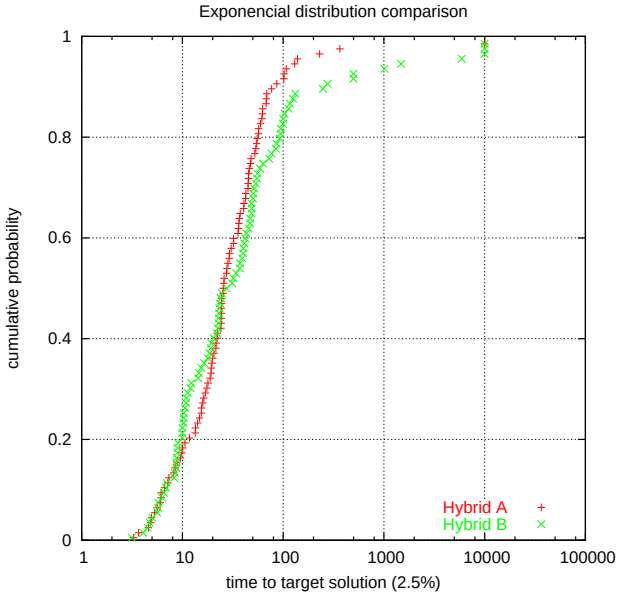

TABLE 4. Average relative percentage increase for Taillard instances. Column CPU 1 indicates the average time (in CPU seconds) spent to find the best solution. Column CPU 2 indicates the average time (in CPU seconds) taken by Hybrid A.

					Hybrid A	
Jobs × Machines	*API*	Best	Worst	StdDev	CPU 1	CPU 2
20 × 5	0.079	0.000	1.138	0.234	0.28	5.06
20 × 10	0.328	0.000	1.194	0.327	5.44	18.74
20 × 20	0.224	0.000	0.826	0.233	8.00	28.64
50 × 5	0.003	0.000	0.071	0.014	15.18	125.24
50 × 10	0.713	0.000	2.407	0.602	206.10	1186.92
50 × 20	1.222	0.324	3.791	0.783	1743.90	4647.40
100 × 5	0.012	0.000	0.285	0.044	129.16	680.38
100 × 10	0.256	0.000	1.263	0.270	322.32	1422.72
100 × 20	1.430	0.702	2.183	0.388	1905.28	4350.28
200 × 10	0.270	0.009	0.754	0.164	498.12	1433.04
200 × 20	1.672	0.782	2.615	0.342	2523.44	4218.04
500 × 20	0.981	0.570	1.459	0.213	2493.02	3894.82
Average	**0.604**					

number of machines increases the effort needed by our algorithms to find a good solution. In part, this could be explained by considering that the makespan for an specific permutation can be calculated by using a function depending on the number of jobs and machines,

TABLE 5. Average relative percentage increase for Taillard instances. Column CPU 1 indicates the average time (in CPU seconds) spent to find the best solution. Column CPU 2 indicates the average time (in CPU seconds) taken by Hybrid B.

					Hybrid B	
Jobs × Machines	*API*	Best	Worst	StdDev	CPU 1	CPU 2
20 × 5	0.058	0.000	0.810	0.167	0.439	4.584
20 × 10	0.275	0.000	2.011	0.382	3.472	14.813
20 × 20	0.231	0.000	0.653	0.186	4.404	28.169
50 × 5	0.019	0.000	0.353	0.071	8.026	93.952
50 × 10	0.700	0.000	1.605	0.485	260.749	1179.363
50 × 20	1.390	0.297	3.787	0.603	1479.574	4108.018
100 × 5	0.006	0.000	0.095	0.020	153.866	687.559
100 × 10	0.258	0.017	0.858	0.256	263.291	1175.743
100 × 20	1.857	0.798	3.624	0.550	1096.554	3575.116
200 × 10	0.257	0.009	0.754	0.184	287.079	1044.240
200 × 20	1.863	0.835	2.766	0.431	1105.303	2129.254
500 × 20	0.902	0.471	1.388	0.203	3149.858	4816.021
Average	**0.651**					

and the time taken on these evaluations explains the need of extra time to reach the same result quality.

Nevertheless, it is quite remarkable to notice that the worst performance of our algorithms happens for the group of instances with 100 jobs and 20 stages (100×20) and for 200×20. And these results are consistent with previous results no matter the method used. The main question to answer is why for 500×20 the algorithm has a better performance. In Taillard's webpage we can find that for 500×20 there are seven out of ten instances with results proved to be optimal solutions, for 200×20 there four out of ten optimal solutions and for the case 100×20 only one optimal result has been found so far and none for 50×20.

Next we discuss some new facts that need review. In 2006, Agarwal et al., (1) discuss an approach to solve the PFSP. In that paper, the authors claim to have found a better upper bound for one instance of Taillard's benchmark by using an adaptive-learning approach (over well known polynomial heuristics). The instance is `ta040` (50 jobs and 5 stages). In the paper, they found a permutation with 2774 as a makespan value, but they did not present the actual permutation. The problem is that in Taillard's website[2] the optimal value for this instance is 2782. Something similar occurs with instance `ta068` (100 jobs and 5 stages) for which we point out two problems. First, the authors incorrectly use an upper bound of 5034, while the correct upper bound, and supposedly also optimal value is 5094. While this could be only a typo, we point out a second problem. The solutions found for two methods presented in that paper have 5082 as a makespan value, which is less than the optimal solution value. Furthermore, the upper bounds for most of the larger instances are exchanged with the corresponding lower bounds. For those cases, the results presented by the authors tend to be better than they actually are. It is important to review these questions to avoid future difficulties for other researchers, especially because these instances are referenced in almost every paper dealing with FSP. For future publications, it seems to be more appropriate to present the actual permutation when a new upper bound or a significant result is reached.

[2] `http://ina2.eivd.ch/Collaborateurs/etd/problemes.dir/problemes.html`

6. Concluding Remarks

In this paper, we discussed the Permutational Flow Shop Problem. This problem has been extensively studied and there are several algorithms published with excellent performance. We presented two implementations of hybrid algorithms based on GRASP, ILS, PR, and a memetic algorithm. In one approach, we use the GRASP-ILS-PR algorithm first maintaining a pool of good and diversified solutions in order to apply path-relinking and to construct a population for the memetic algorithm. In the second approach, we use the memetic algorithm to improve the pool of elite solutions every time the GRASP-ILS-PR algorithm reaches a state of stagnation for a specific number of iterations. The memetic algorithm uses a crossover based on path-relinking (PX) that was shown to be very effective. The best performance is obtained using Hybrid A, when we allow the GRASP-ILS-PR to perform a large search almost 95% of the time and then the memetic algorithm try to obtain a better solution using the output of the previous one.

The results are competitive, showing that for the hardest instances of Taillard's benchmark, i.e. the ones with 100 jobs and 20 stages, the heuristics can find a solution no worse than 2.5% of the best known upper bound in less than 80 seconds with 80% probability.

We also discussed some characteristics about the difficulty of the benchmark and some new results presented in a recent paper that need to be reviewed.

This paper is an important step in the direction of a more ambitious project in which we consider a *real-world* scheduling problem. The next step in our research is the consideration of sequence dependent setups and due dates in some or all stages. We are also considering a multi-objective approach, trying to obtain a flexible algorithm that could face different practical circumstances. We believe that in these more complicated cases the use of these kinds of hybrid approaches will be most advantageous.

Acknowledgement

This research was supported by CNPq, NSF, and U.S. Air Force Grants.

References

[1] A. Agarwal, S. Colak, and E. Eryarsoy. Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach. *European Journal of Operational Research*, 169(3):801–815, 2006.

[2] R. K. Ahuja, J. B. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assigment problem. *Computers & Operations Research*, 27(10):917–934, 2000.

[3] R.M. Aiex, S. Binato, and M.G.C. Resende. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 29:393–430, 2003.

[4] R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 8:343–373, 2002.

[5] R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. TTTPLOTS: A perl program to create time-to-target plots. Technical report, AT&T Labs,Technical Report TD-6HT7EL, 2005. To appear in *Optimization Letters*.

[6] K. R. Baker. Requirements planning. In *Handbooks in OR & MS*, editors S C Graves et al.1993.

[7] M. Ben-Daya and M. Al-Fawzan. A tabu search approach for the flow shop scheduling problem. *European Journal of Operational Research*, 109(1):88–95, 1998.

[8] E. Coffman. *Computer and Job-Shop Scheduling Theory*. Wiley, New York, 1976.

[9] D. G. Dannenbring. An evaluation of flow shop sequencing heuristics. *Management Science*, 23(11):1174–1182, 1977.
[10] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.
[11] P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys in metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.
[12] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Nav. Res. Log. Quarterly*, 1:61–68, 1954.
[13] A.H.G. Rinnooy Kan. *Machine scheduling problems: classification, complexity and computations*. The Hague : Nijhoff, 1976.
[14] H. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of metaheuristics*. Kluwer, 2003.
[15] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998.
[16] M. Nawaz, E.E. Enscore, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.
[17] F.A. Ogbu and D.K. Smith. The application of the simulated annealing to the solution of the n/m/cmax flow shop problem. *computers & Operations Research*, 17(3):243–253, 1990.
[18] C.R. Reeves. A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1):5–13, 1995.
[19] M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2003.
[20] M.G.C. Resende and C.C. Ribeiro. GRASP with path-relinking: Recents advances and aplications. *Metaheuristics: Progress as Real Problem Solvers. Ibakari and Nonobe and Yaguira (eds). Springer*, pages 29–63, 2005.
[21] R. Ruiz and C. Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494, 2005.
[22] R. Ruiz, C. Maroto, and J. Alcaraz. New genetic algorithms for the permutational flowshop scheduling problem. *MIC2003: The Fifth Metaheuristics International Conference*, pages 63.1–63.8, 2003.
[23] R. Ruiz, C. Maroto, and J. Alcaraz. Two new robust genetic algorithms for the flowshop scheduling problem. *OMEGA, The International Journal of Management Science*, 34:461–476, 2006.
[24] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, Available on-line, 2006.
[25] T. Stützle. Applying iterated local search to the permutation flow shop problem. Technical report, TU Darmstadt, AIDA-98-04, FG Intellektik, 1998.
[26] É. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74, 1990.
[27] É. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

(Martín G. Ravetti) DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO, UNIVERSIDADE FEDERAL DE MINAS GERAIS, BELO HORIZONTE, MG, BRAZIL.
E-mail address: martin@dcc.ufmg.br

(Fabiola G. Nakamura) DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO, UNIVERSIDADE FEDERAL DE MINAS GERAIS, BELO HORIZONTE, MG, BRAZIL.
E-mail address: fgnaka@dcc.ufmg.br

(Claudio N. Meneses) DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING, UNIVERSITY OF FLORIDA, GAINESVILLE, FL, USA.
E-mail address: claudio@ufl.edu

(Mauricio G. C. Resende) ALGORITHMS AND OPTIMIZATION RESEARCH DEPARTMENT, AT&T LABS RESEARCH, 180 PARK AVENUE, ROOM C241, FLORHAM PARK, NJ 07932 USA.
E-mail address: mgcr@research.att.com

(Geraldo R. Mateus) DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO, UNIVERSIDADE FEDERAL DE MINAS GERAIS, BELO HORIZONTE, MG, BRAZIL.
E-mail address: mateus@dcc.ufmg.br

(Panos M. Pardalos) DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING, UNIVERSITY OF FLORIDA, GAINESVILLE, FL, USA.
E-mail address: pardalos@ufl.edu