Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

A SIMULATION-BASED APPROACH FOR SOLVING THE FLOWSHOP PROBLEM

Angel A. Juan

Technical University of Catalonia – IEMAE
School of Building Construction of Barcelona
Barcelona, 08028, SPAIN

Rubén Ruíz

Universidad Politécnica de Valencia
Grupo Sistemas Optimización Aplicada – ITI
Valencia, 46022, SPAIN

Helena R. Lourenço

Universitat Pompeu Fabra
Department of Economics and Business
Barcelona 08005, SPAIN

Manuel Mateo

Universidad Politécnica de Cataluña
Departamento de Organización de Empresas
Barcelona, 08028, SPAIN

Dragos Ionescu

Open University of Catalonia – IN3
Computer Science and Telecommunication Studies
Barcelona, 08018, SPAIN

ABSTRACT

A simulation-based algorithm for the Permutation Flowshop Sequencing Problem (PFSP) is presented. The algorithm uses Monte Carlo Simulation and a discrete version of the triangular distribution to incorporate a randomness criterion in the classical Nawaz, Enscore, and Ham (NEH) heuristic and starts an iterative process in order to obtain a set of alternative solutions to the PFSP. Thus, a random but biased local search of the space of solutions is performed, and a list of "good alternative solutions" is obtained. We can then consider several properties per solution other than the makespan, such as balanced idle times among machines, number of completed jobs at a given target time, etc. This allows the decision-maker to consider multiple solution characteristics apart from those defined by the aprioristic objective function. Therefore, our methodology provides flexibility during the sequence selection process, which may help to improve the scheduling process. Several tests have been performed to discuss the effectiveness of this approach. The results obtained so far are promising enough to encourage further developments and improvements on the algorithm and its applications in real-life scenarios. In particular, Multi-Agent Simulation is proposed as a promising technique to be explored in future works.

1 INTRODUCTION

The Permutation Flowshop Sequencing Problem (PFSP) is a well-known scheduling problem that can be described as follows: a set J of n independent jobs has to be processed on a set M of m independent machines. Every job $j \in J$ requires a given fixed processing time $p_{ij} \geq 0$ on every machine $i \in M$. Each machine can execute at most one job at a time, and it is assumed that the all jobs are processed by

the machines in the same order. The classical goal is to find a sequence for processing the jobs in the shop so that a given criterion is optimized. The criterion most commonly used is the minimization of the maximum completion time (makespan) denoted by $C_{\max}$. Figure 1 illustrates this problem for a simple case of $n=3$ jobs and $m=3$ machines.

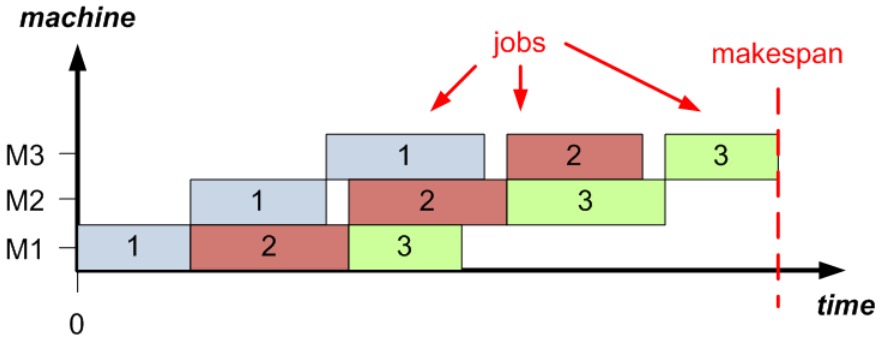

Figure 1: Flowshop Sequencing Problem

The described problem is usually denoted as $Fm|prmu|C_{max}$ and it is a combinatorial problem with $n!$ possible sequences which, in general, is NP-complete (Rinnooy Kan, 1976). Similar to what has happened with other combinatorial problems, a large number of different approaches have been developed to deal with the PFSP. These approaches range from the use of pure optimization methods, such as mixed integer programming, for solving small-sized problems to the use of heuristics and metaheuristics that provide near-optimal solutions for medium and large-sized problems. Most of these methods focus on minimizing makespan. However, utility functions of decision-makers are difficult to model and, frequently, criteria other than total time employed to process the list of jobs should be also considered in real-life scenarios. For that reason, there is a need for more flexible methods able to provide a large set of alternative near-optimal solutions with different properties, so that decision-makers can choose among different alternative solutions according to their specific necessities and preferences.

In Cordeau et al (2002), four attributes are stated for a good heuristic: accuracy, speed, simplicity, and flexibility. On the one hand, Ruiz and Stutzle (2007) presented a simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, which can be considered very accurate and obtains solutions in very short time. On the other hand, the procedures designed following a GRASP (Greedy Randomize Adaptative Search Procedure), see some algorithms for example in Festa and Resende (2009b), can be qualified as very simple and flexible. Our proposal will be built trying to balance in an efficient way the four attributes, i.e. with a simple and flexible algorithm, fast enough and with an important degree of accuracy. In fact, the presented algorithm is close to GRASP as randomness is introduced in the search of good solutions and simulation can be seen similar to the repetition done at each iteration.

In some recent works, Juan et al. (2009a, 2009b, 2010) described the application of simulation techniques to solve Vehicle Routing Problems (VRP). They commented on the convenience of using similar approaches for combinatorial problems other than the Capacitated VRP (CVRP): "it is convenient to highlight that the introduced methodology can be used beyond the CVRP scenario… similar hybrid algorithms based on the combination of Monte Carlo simulation with already existing heuristics can be developed for other routing problems… and, in general, for other combinatorial optimization problems". Following their ideas, this article presents a hybrid approach that uses MCS to randomize the NEH classical heuristic for dealing with the PFSP. The paper also discusses how Multi-Agent Simulation (MAS) could be used to further improve the proposed methodology.

2 LITERATURE REVIEW

A large number of heuristics and metaheuristics have been proposed to solve the PFSP, mainly because of the difficulty encountered by exact methods to solve medium or large instances. As explained before, most existing approaches focus on the objective of minimizing makespan. Johnson (1954) proposed a simple heuristic to obtain optimal sequences for the PFSP with two machines. Campbell, Dudek, and Smith (1970) developed the so-called CDS heuristic for solving the PFSP with more than two machines. Dannenbring (1977) also proposed several constructive and improvement heuristics for the general problem. Nawaz, Enscore, and Ham (1983) introduced the NEH heuristic, which is commonly considered as the best performing heuristic for the PFSP. Basically, what the NEH heuristic proposes is to calculate the total processing time required for each job (i.e., the total time each jobs requires to be processed by the set of machines), and then to create an "efficiency list" of jobs sorted in a descending order according to this total processing time. At each step, the job at the top of the efficiency list is selected and used to construct the solution, that is: the "common sense" rule is to select first those jobs with the highest total processing time. Once selected, a job is inserted in the sorted set of jobs that are configuring the ongoing solution. The exact position that the selected job will occupy in that ongoing solution is given by the minimizing makespan criterion. Taillard (1990) introduced a data structure that reduces the NEH complexity. The makespan of all n possible insertions point in this structure can be calculated by three matrices: e (heads), q (tails) and f (heads plus processing times). These three matrices avoid to explicitly calculate the makespan. In our approach we use this data structure, since it is rather straightforward to implement. Other interesting heuristics are those from Suliman (2000) or Framinan and Leisten (2003), which also consider several extensions of NEH when facing objectives other than makespan.

Different metaheuristic approaches have been also proposed for the PFSP. Osman and Potts (1989) used Simulated Annealing. Widmer and Hertz (1989) proposed a Tabu Search algorithm known as SPIRIT. Other Tabu Search algorithms, which make use of the NEH heuristic, were introduced by Taillard (1990) and Reeves (1993). Chen, Vempati, and Aljaber (1995), Reeves (1995) and Aldowaisan and Allahvedi (2003) proposed the use of Genetic Algorithms for solving the PFSP which were also based on the NEH heuristic. As examples of other works which also rely on the use of the NEH heuristic we can cite the Ant Colony Optimization algorithm of Chandrasekharan and Ziegler (2004).

All the aforementioned work has in common that the algorithms proposed are easy to code and therefore the results can be reproduced without too much difficulty. In addition, many of the above algorithms can be adapted to other more realistic flowshop environments (Ruiz and Maroto 2005). Additionally, there are other highly elaborated hybrid techniques for solving the PFSP. However, as Ruiz and Stützle (2007) point out, "they are very sophisticated and an arduous coding task is necessary for their implementation." In other words, it is unlikely that they can be used for solving realistic scenarios without direct support from the researchers that developed them.

Regarding works facing a multiobjective function in the framework of the Permutation Flow Shop Scheduling, we can classify them into three main groups:

1. A single unified objective function: Rajendran (1995) wanted to minimize the equal-weighted sum of normalized makespan, total flow time and machine idle time. Genetic Algorithms have been a usual technique in this group. Cavalieri and Gaiardelli (1998) used a non-linear aggregation function to convert makespan and total tardiness into a single value and this value was treated as the fitness for the Genetic Algorithm.
2. A main criterion and another secondary criterion: Neppalli, Chen, and Gupta (1996) consider the objective of minimizing the total flow time subject to obtaining the optimal makespan for a two-stage flow shop scheduling problem. This contains the view closer to our resolution procedures.
3. The biobjective case reflected in the search of solutions in the Pareto front structure: For example, Chiang, Cheng, and Fu (2009) faced the scheduling of the permutation flow shop with minimization of makespan and total flow time as the objectives through a memetic algorithm (MA) which

searched for the set of non-dominated solutions. As they comment in the paper, "entering the new century, there was a rapid growth of multiobjective evolutionary algorithms (MOEA)".

3 REVIEW OF GRASP ALGORITHMS

Since the approach presented in this paper has some similarities with GRASP metaheuristics, this family of algorithms is briefly described next. For a recent review of this general methodology the reader is referred to Festa and Resende (2009a, 2009b). Greedy Randomized Adaptive Search Procedure (GRASP) is a multi-start method designed to solve hard combinatorial optimization problems (Feo and Resende, 1995). The basic methodology consists in two phases: (a) a constructive phase that builds a good but not necessarily locally optimal solution, and (b) a second phase which consists in a local search procedure. The two phases are repeated until a stopping criterion is reached, keeping track of the best solution found over all the search. The constructive phase builds step by step adding an element to a partial solution following a greedy function. The selection of the element to be added in each iteration is not deterministic but subjected to a randomization process. In that way, the repetition of both the phases leads to different solutions. The randomization process is usually controlled by a parameter δ, that in the simplest versions of GRASP is fixed along the execution of the algorithm.

A particularly interesting GRASP is the so called Reactive GRASP (Prais and Ribeiro, 2000). In this version of the methodology, the parameter δ is not fixed along the running of the algorithm, but it is selected randomly from a set of discrete values. Initially, all values have the same probability of being chosen. In the iterative process we keep the value of the solutions obtained for each value of δ. After a certain number of iterations the probabilities are modified. Those corresponding to values of δ which have produced good solutions are increased and, conversely, those corresponding to values producing low quality solutions are decreased.

4 OUR SIMULATION-BASED APPROACH

The approach presented in this paper aims to provide a simple probabilistic algorithm that will improve the results obtained with the NEH heuristic in a few iterations, and then use the refined solution to produce an answer that is very close (identical for small instances) to the best-known solution. The main goal is to develop a parameter-free algorithm in order to avoid complex or time-consuming fine-tuning processes. As explained before, a second goal of our approach is to provide not only a good solution, but also a set of alternative solutions.

To meet these goals, we combine Monte Carlo Simulation (MCS) techniques with the NEH heuristic. In particular, our approach introduces a special random behavior within the NEH heuristic and then starts an iterative process. Basically, we initiate a search process inside the space of feasible solutions. Each such solution will consist of a list of jobs sorted in some order. As explained before, the NEH heuristic is an iterative algorithm which employs a list of jobs sorted by their total completion time on all the machines to construct a solution for the PFSP. At each step of this iterative process, the NEH removes the job which is at the top of that list (with maximum completion time) and adds it to a new list at the position that results in the best partial solution with respect to makespan. As a result, the NEH provides a "common sense" deterministic solution, by trying to schedule the most demanding jobs first. Our method, instead, assigns a probability of selecting each job in the list. According to our design, this probability should be coherent with the total time that each job needs to be processed by all the machines, i.e., jobs with higher total times will be more likely to be selected from the list before those with lower total times. Finally, the selection process should be done without introducing any parameters in the methodology –otherwise, it would be necessary to perform fine-tuning processes, which tend to be non-trivial and time-consuming. To satisfy all these requirements, we employ a discrete version of a triangular distribution during the solution-construction process: each time a new job has to be selected from the list, a triangular distribution that assigns linearly diminishing probabilities to each eligible job according its corresponding total-processing-time value is employed. That way, jobs with higher processing times are always

more likely to be selected from the list first, but the probabilities assigned are variable and they depend upon the concrete distribution selected at each step. By iterating this procedure, an oriented random search process is started. We call the resulting job orderings Randomized NEH solutions. Notice that this general approach has many similarities with the GRASP procedures described before. To some extent, our approach could be considered as part of the large family of Hybrid GRASP algorithms. Even if the results included in this paper are based on the use of a discrete triangular distribution, other distributions such as the Log-normal or Zipf could also be considered.

After we find a base solution that is better than the one given by the NEH heuristic, we try to further optimize it by applying a local search. The idea is to pick at random and without repetition a position from the list and find the best place for the corresponding job. We do this n times, where n is the number of jobs, and we use Taillard accelerations at each step to speed up the execution. The local search process is repeated as long as we keep getting better arrangements of the jobs. Note that this second stage of our algorithm will never output a solution that is worse than the base solution. The completion time for the configuration of jobs obtained at the end of this process is thus always less than the NEH makespan. However, if we would stop our algorithm at this point, we would have a high chance of ending in the vicinity of a local minimum that can be still far away from the optimal solution. To avoid this problem, we restart the entire process, compute a new base solution, and improve it with the local search procedure as long as the total running time is less than a certain threshold –e.g. 0.030 seconds $*$ (number of jobs) $*$ (number of machines) for our implementation. In the multi-start process we keep track of the best solution seen so far and we update it whenever necessary. By starting our search at different locations in the solution space we also increase considerably the chances of finding configurations that are close to the actual optimum. The main steps of our algorithm are summarized next (Figure 2):

1. Given a PFSP instance, construct the corresponding data model and use the classical NEH algorithm to solve it. Assign the NEH solution to the variable holding the best solution seen so far.
2. Choose a probability distribution (e.g. a triangular) for adding random behavior to the algorithm.
3. Start an iterative process to generate Randomized NEH (RNEH) solutions. Stop once you find a solution that outperforms the one provided by the NEH algorithm, i.e. a base solution.
4. Keep applying a local search to the base solution computed in step 3 as long as you get improvements.
5. Compare the configuration obtained in step 4 with the best solution seen so far, and update the latter if necessary.
6. Go back to step 2 and restart the process if time permits.

To increase the probability of getting a good solution we run each instance with 15 different seeds. The results are presented in the next section.

5 EXPERIMENTAL RESULTS

The methodology described in this paper has been implemented as a Java application. Java was chosen here since, being an object-oriented language, it facilitates the rapid development of a prototype. An Intel Xeon at 2.0 GHz and 4 GB RAM, was used to perform all tests, which were run directly on the Eclipse IDE for Java over Windows 7.

We tested the first 60 benchmark instances from Taillard (1993) that can be found at Taillard (2010). For each tested instance, we also considered its associated best-known solution (BKS) as reported in Zobolas, Tarantilis, and Ioannou (2009). Table 1 shows the results for the first 30 Taillard's instances, which consider cases with 20 jobs. The first 6 columns offer general information: the order of the instance, the number of jobs, the number of machines, the makespan of the BKS, the makespan of the NEH solution, and the gap between the two. The last 6 columns contain data describing the performance of our algorithm. Note that we run each instance with 15 different seeds, so we provide in the table an average cost (AvCost), an average time (AvTime) and an average gap with respect to the cost of the BKS for these runs. The next two columns give information about the makespan of the best solution we find (MinCost),

and its associated running time (MinTime). Finally, the last column contains the gap between the cost of our solution and that of the BKS.

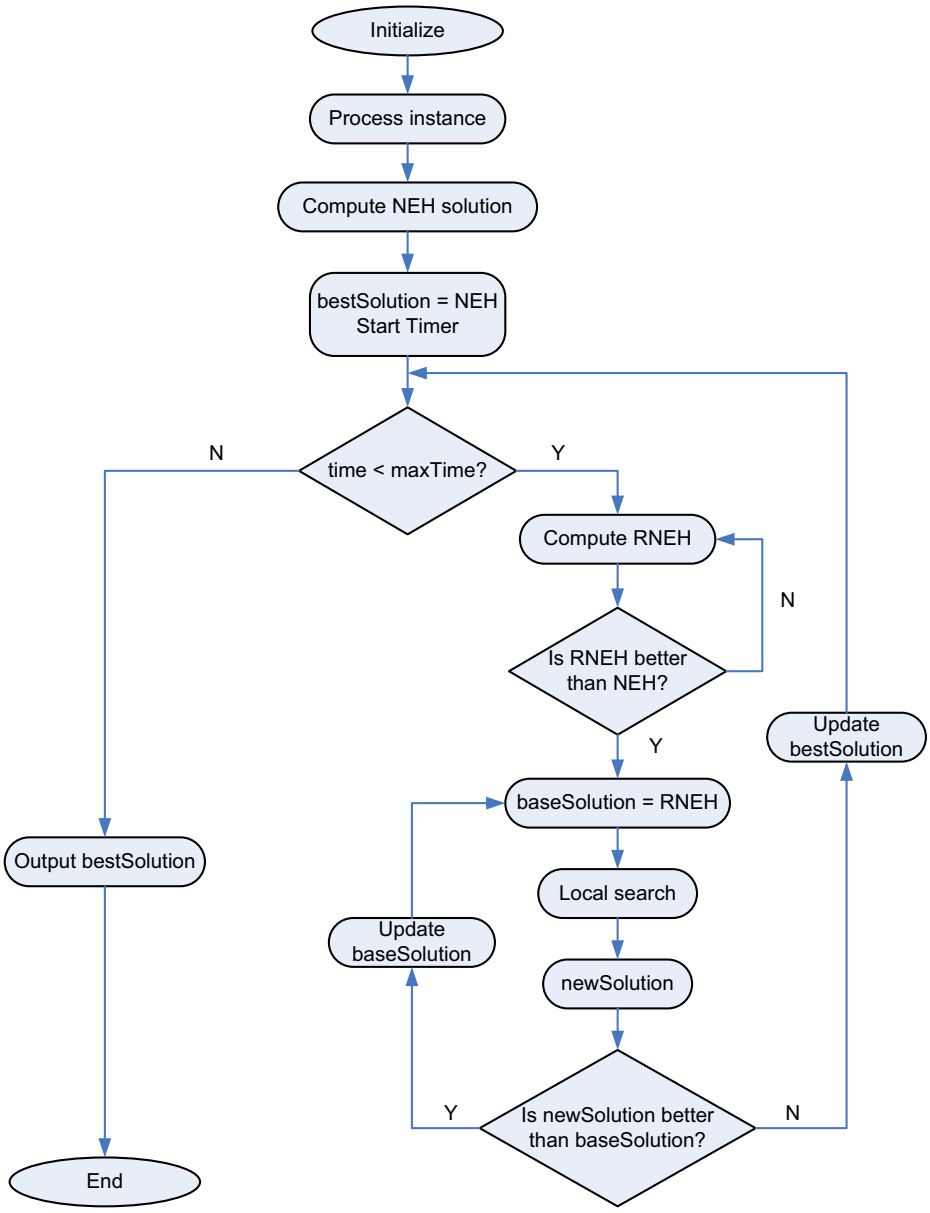

Figure 2: Flow diagram for our algorithm

Notice the significant reduction in the average gap with respect to the best-known solution, from 3.88% (NEH) to 0.00% (our algorithm). As can be seen in Table 1, for the first 30 Taillard's instances our methodology is able to provide a solution that is as good as the best-known solution.

Table 2 shows results for the Taillard's instances with 50 jobs. These are, of course, larger instances but again our methodology provides a noticeable reduction in the average gap (from 4.15% to 0.82%) in a reasonable amount of computing time (5.5 seconds on the average). The times can be easily improved by just coding the algorithm in C/C++ instead of Java (being an interpreted language, Java does not offer the best execution-speed performance) or simply by using a more powerful computer. We are currently in

the process of developing an optimized C/C++ version of the code and, according to our previous experiences with similar algorithms, we expect to significantly reduce the CPU times for most of these 60 instances.

Table 1: Results for Taillard's instances 1 to 30 (20 jobs)

Taillard's Instance	# Jobs	# Machines	BKS	NEH Solution	Gap BKS-NEH	AvCost	AvTime	AvGap	MinCost	MinTime	MinGap
1	20	5	1278	1286	0.63%	1278.0	0.0050	0.00%	1278	0.0002	0.00%
2	20	5	1359	1365	0.44%	1359.0	0.1712	0.00%	1359	0.0134	0.00%
3	20	5	1081	1159	7.22%	1081.0	0.0272	0.00%	1081	0.0011	0.00%
4	20	5	1293	1325	2.47%	1293.0	0.2897	0.00%	1293	0.0343	0.00%
5	20	5	1235	1305	5.67%	1235.0	0.0366	0.00%	1235	0.0088	0.00%
6	20	5	1195	1228	2.76%	1195.0	0.0049	0.00%	1195	0.0008	0.00%
7	20	5	1234	1278	3.57%	1238.0	0.5076	0.32%	1234	2.2094	0.00%
8	20	5	1206	1223	1.41%	1206.0	0.0059	0.00%	1206	0.0008	0.00%
9	20	5	1230	1291	4.96%	1230.0	0.0119	0.00%	1230	0.0012	0.00%
10	20	5	1108	1151	3.88%	1108.0	0.0154	0.00%	1108	0.0009	0.00%
11	20	10	1582	1680	6.19%	1582.1	1.4686	0.00%	1582	0.1500	0.00%
12	20	10	1659	1729	4.22%	1659.1	1.7272	0.00%	1659	0.0756	0.00%
13	20	10	1496	1557	4.08%	1496.5	2.3979	0.03%	1496	0.1155	0.00%
14	20	10	1377	1439	4.50%	1377.3	2.3866	0.02%	1377	0.0378	0.00%
15	20	10	1419	1502	5.85%	1419.0	0.1121	0.00%	1419	0.0175	0.00%
16	20	10	1397	1453	4.01%	1397.0	0.7522	0.00%	1397	0.0325	0.00%
17	20	10	1484	1562	5.26%	1484.0	0.2539	0.00%	1484	0.0053	0.00%
18	20	10	1538	1609	4.62%	1542.6	2.2702	0.30%	1538	0.9220	0.00%
19	20	10	1593	1647	3.39%	1594.5	2.5745	0.10%	1593	0.3226	0.00%
20	20	10	1591	1653	3.90%	1591.2	1.8337	0.01%	1591	0.1282	0.00%
21	20	20	2297	2410	4.92%	2297.3	4.3805	0.01%	2297	1.8888	0.00%
22	20	20	2099	2150	2.43%	2099.0	1.4351	0.00%	2099	0.0608	0.00%
23	20	20	2326	2411	3.65%	2327.6	5.3052	0.07%	2326	0.4416	0.00%
24	20	20	2223	2262	1.75%	2223.0	0.0640	0.00%	2223	0.0134	0.00%
25	20	20	2291	2397	4.63%	2292.8	2.5181	0.08%	2291	0.0097	0.00%
26	20	20	2226	2349	5.53%	2226.8	4.7572	0.04%	2226	0.7903	0.00%
27	20	20	2273	2362	3.92%	2273.0	3.7247	0.00%	2273	0.5397	0.00%
28	20	20	2200	2249	2.23%	2200.0	4.1436	0.00%	2200	0.1868	0.00%
29	20	20	2237	2320	3.71%	2237.0	1.0461	0.00%	2237	0.0858	0.00%
30	20	20	2178	2277	4.55%	2178.9	4.6251	0.04%	2178	1.5517	0.00%
Averages					*3.88%*		*1.6284*	*0.03%*		*0.3215*	*0.00%*

As described before, our approach makes use of an iterative process to generate a set of random feasible solutions. According to the experimental tests carried out, each iteration is completed in just a few milliseconds by using a standard computer. By construction, odds are that the generated solution outperforms the one given by the NEH heuristic. This means that our approach provides, after a few iterations, a feasible solution which outperforms the NEH heuristic as regards makespan. Moreover, as verified by testing, several alternative solutions improving the NEH can be obtained after some more iterations (which take significantly more CPU time), each of them having different attributes regarding criteria such as balanced idle times among machines, number of completed jobs at a given target time, etc.

Another important point to consider here is the simplicity of the presented methodology. In effect, our algorithm needs little instantiation and does not require any fine-tuning or set-up processes. This is

quite interesting in our opinion, since as it was noticed before, some of the most efficient metaheuristics are not used in practice because of the difficulties they present when trying to implement them. On the contrary, simple hybrid approaches like the one introduced here tend to be more flexible and, therefore, they seem more appropriate to deal with real restrictions and dynamic work conditions.

Table 2: Results for Taillard's instances 31 to 60 (50 jobs)

Taillard's Instance	# Jobs	# Machines	BKS	NEH Solution	Gap BKS-NEH	AvCost	AvTime	AvGap	MinCost	MinTime	MinGap
31	50	5	2724	2733	0.33%	2724.0	0.0080	0.00%	2724	0.0022	0.00%
32	50	5	2834	2843	0.32%	2835.6	1.0068	0.06%	2834	1.0319	0.00%
33	50	5	2621	2640	0.72%	2621.0	0.1180	0.00%	2621	0.0054	0.00%
34	50	5	2751	2782	1.13%	2751.0	0.6044	0.00%	2751	0.0102	0.00%
35	50	5	2863	2868	0.17%	2863.0	0.2503	0.00%	2863	0.0059	0.00%
36	50	5	2829	2850	0.74%	2829.0	0.2978	0.00%	2829	0.0270	0.00%
37	50	5	2725	2758	1.21%	2725.0	0.1352	0.00%	2725	0.0235	0.00%
38	50	5	2683	2721	1.42%	2683.0	0.0334	0.00%	2683	0.0063	0.00%
39	50	5	2552	2576	0.94%	2552.1	2.4279	0.00%	2552	0.3099	0.00%
40	50	5	2782	2790	0.29%	2782.0	0.0167	0.00%	2782	0.0026	0.00%
41	50	10	2991	3135	4.81%	3035.1	9.1256	1.47%	3028	10.2698	1.24%
42	50	10	2867	3032	5.76%	2914.6	7.7866	1.66%	2911	3.5524	1.53%
43	50	10	2839	2986	5.18%	2882.2	7.9055	1.52%	2873	2.5159	1.20%
44	50	10	3063	3198	4.41%	3069.0	2.4458	0.20%	3063	8.0892	0.00%
45	50	10	2976	3160	6.18%	3015.5	6.1041	1.33%	3001	8.2741	0.84%
46	50	10	3006	3178	5.72%	3032.9	6.5275	0.89%	3020	11.5064	0.47%
47	50	10	3093	3277	5.95%	3127.3	7.9190	1.11%	3124	0.8654	1.00%
48	50	10	3037	3123	2.83%	3044.5	5.4414	0.25%	3042	0.3796	0.16%
49	50	10	2897	3002	3.62%	2917.1	4.6210	0.69%	2907	7.1431	0.35%
50	50	10	3065	3257	6.26%	3114.1	6.4705	1.60%	3100	5.6446	1.14%
51	50	20	3850	4082	6.03%	3929.3	15.2113	2.14%	3921	21.2587	1.92%
52	50	20	3704	3921	5.86%	3773.9	13.7315	1.89%	3755	16.4208	1.38%
53	50	20	3640	3927	7.88%	3729.2	16.1167	2.45%	3719	1.7320	2.17%
54	50	20	3720	3969	6.69%	3796.2	17.7403	2.08%	3781	11.9047	1.67%
55	50	20	3610	3835	6.23%	3683.4	11.4236	2.03%	3672	6.4948	1.72%
56	50	20	3681	3914	6.33%	3753.9	16.8949	2.03%	3746	18.7676	1.82%
57	50	20	3704	3952	6.70%	3776.4	13.9928	1.95%	3743	8.3589	1.05%
58	50	20	3691	3938	6.69%	3778.6	11.4161	2.37%	3768	5.9899	2.09%
59	50	20	3743	3952	5.58%	3808.7	15.6581	1.81%	3796	12.6947	1.47%
60	50	20	3756	4079	8.60%	3825.0	12.2771	1.84%	3806	0.3988	1.33%
Averages					*4.15%*		*7.1236*	*1.05%*		*5.4562*	*0.82%*

6 FUTURE WORK

The algorithm presented here can be easily parallelized by splitting the random-number-generation sequence in different streams and using each stream in different threads or CPUs. This can be an interesting field to explore in future works, given the current trend in multi-core processors and parallel computing.

Offering competitive solutions to complex problems in real time and without adjustments beforehand still presents a challenge. In spite of this, it has been empirically observed that it is possible to significantly reduce the execution time that the algorithm needs to obtain good solutions, depending on the seed that is chosen for the pseudo-random number generator. There are new processor design paradigms based on gaining computation capacity through the parallel execution of multiple processes and threads (multi-

core). Following this concept, new, affordable Graphic Processing Units (GPUs) have recently been introduced on the market that offer the capacity of executing hundreds –or even thousands– of threads concurrently.

Accordingly, our current goal is to implement a multi-threaded version of the previously described algorithm in the language C/C++. This will allow us to execute multiple instances of the algorithm at the same time, each with a different seed. Each of these instances can be considered as an individual agent that is searching the solution space by using a Multi-Agent Simulation approach. In other words, the idea is that each of these multiple agents will start to search in a different region of the solution space (by using different seeds) (Figure 3).

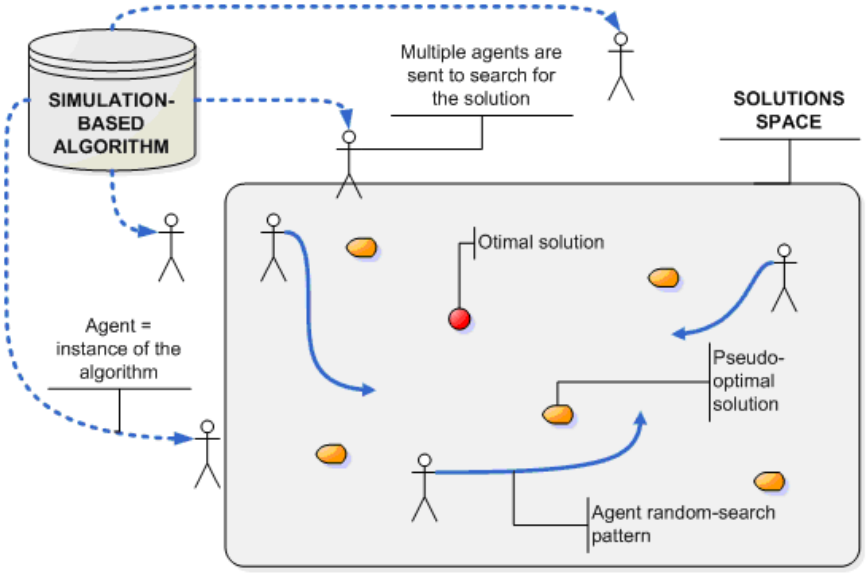

Figure 3: Designing a Multi-Agent Simulation approach

Our hypothesis here is that a multi-threaded version of the algorithm can provide very competitive results to the PFSP problem for most benchmarks. At the same time, using C will facilitate translating the code to CUDA, an extension of C that can use all of the parallel processing power of the new GPUs that are commercially available now.

7 CONCLUSIONS

In this paper a simple probabilistic methodology for solving the Permutation Flowshop Sequencing Problem (PFSP) has been presented. This methodology, which does not require any particular fine-tuning or configuration process, combines the classical NEH heuristic with Monte Carlo simulation using a triangular distribution.

Results show that our methodology is able to improve the NEH heuristic in just a few iterations. Moreover, being a constructive approach, it can generate several alternative good solutions in a reasonable time-period. The paper also discusses some ongoing research on the use of C/C++ versions of the code to significantly reduce computation times as well as on the use of multi-agent simulation techniques to accelerate even further the methodology when applied to very large problems.

ACKNOWLEDGEMENTS

This work has been partially supported by the Spanish Ministry of Science and Innovation (grants TRA2010-21644-C03 and DPI2007-61371), by the Catalan Department of Universities, Research & In-

formation Society (grant 2009 CTP 00007) and by the HAROSA Knowledge Community of the Internet Interdisciplinary Institute (http://dpcs.uoc.edu).

REFERENCES

Aldowaisan, T., and A. Allahvedi. 2003. New heuristics for no-wait flowshops to minimize makespan. *Computers and Operations Research* 30(8):1219-1231.

Campbell, H.G., R.A. Dudek, and M.L. Smith. 1970. A heuristic algorithm for the n job, m machine sequencing problem. *Management Science* 16:B630-B637.

Cavalieri, S., and P. Gaiardelli. 1998. Hybrid Genetic Algorithms for a Multiple-objective Scheduling Problem. *Journal of Intelligent Manufacturing* 9:361-367.

Chandrasekharan, R., and H. Ziegler. 2004. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research* 155(2):426-438.

Chen, C.L., V.S. Vempati, and N. Aljaber. 1995. An application of genetic algorithms for flow shop problems. *European Journal of Operational Research* 80(2):389-396.

Chiang, T.-C., H.-C. Cheng, and L.-C. Fu. 2009. Multiobjective Permutation Flow Shop Scheduling Using a Memetic Algorithm with an NEH-Based Local Search. In *Emerging Intelligent Computing Technology and Applications*, ed. D.-S. Huang, K.-H. Jo, H.-H. Lee, H.-J. Kang, and V. Bevilacqua, 813–825. Berlin / Heidelberg: Springer.

Cordeau, J.-F., M. Gendreau, G. Laporte, J.-Y. Potvin, and F. Semet. 2002. A guide to vehicle routing heuristics. *Journal of the Operational Research Society* 53:512-522.

Dannenbring, D.G. 1977. An evaluation of flowshop sequence heuristics. *Management Science* 23:1174-1182.

Feo, T. A., and M.G.C. Resende. 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization* 6, 109-133.

Festa, P., and M.G.C. Resende. 2009a. An annotated bibliography of GRASP—Part I: algorithms. *International Transactions in Operational Research* 16:1-24.

Festa, P., and M.G.C. Resende. 2009b. An annotated bibliography of GRASP—Part II: applications. *International Transactions in Operational Research* 16:131–172.

Framinan, J.M., and R. Leisten. 2003. An efficient constructive heuristic for flowtime minimisation in permutation flow shops. *OMEGA* 31:311-317.

Juan, A., J. Faulin, D. Riera, D. Masip, and J. Jorba. 2009a. A Simulation-based Methodology to assist Decision-makers in real Vehicle Routing Problems. In *Proceedings of the 11th International Conference on Enterprise Information Systems*, ed. J. Cordeiro and F. Joaquim, 212-217.

Juan, A., J. Faulin, R. Ruiz, B. Barrios, and S. Caballe. 2010. The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem. *Applied Soft Computing* 10(1):215-224.

Juan, A., J. Faulin, R. Ruiz, B. Barrios, M. Gilibert, and X. Vilajosana. 2009b. Using oriented random search to provide a set of alternative solutions to the capacitated vehicle routing problem. In *Operations Research and Cyber-Infrastructure*, 331-346. Springer.

Johnson, S.M. 1954. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly* 1:61-68.

Nawaz, M., E.E. Enscore, and I. Ham. 1983. A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. *OMEGA* 11:91-95.

Neppalli, V.R., C.L. Chen, and J.N.D. Gupta. 1996. Genetic Algorithms for the two-stage bicriteria flowshop problem. *European Journal of Operational Research* 95:356-373.

Osman, L., and C. Potts. 1989. Simulated annealing for permutation flow-shop scheduling. *OMEGA* 17(6):551-557.

Prais, M., and C.C. Ribeiro. 2000. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing* 12:164-176.

Rajendran, C. 1995. Heuristics for Scheduling in Flowshop with Multiple Objectives. *European Journal of Operational Research* 83:540–555.

Reeves, C.R. 1993. Improving the efficiency of tabu search for machine scheduling problems. *Journal of the Operational Research Society* 44(4):375-382.

Reeves, C.R. 1995. A genetic algorithm for flowshop sequencing. *Computers and Operations Research* 22(1):5-13.

Rinnooy Kan, A.H.G. 1976. *Machine Scheduling Problems: Classification, Complexity and Computations*. Springer.

Ruiz, R., and C. Maroto. 2005. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research* 165:479-494.

Ruiz, R., and T. Stützle. 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research* 177:2033-2049.

Suliman, S. 2000. A two-phase heuristic approach to the permutation flow-shop scheduling problem. *International Journal of Production Economics* 65 (1-3):143-152.

Taillard, E. 1990. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research* 47:65-74.

Taillard, E. 1993. Benchmarks for basic scheduling problems. *European Journal of Operations Research* 64:278-285.

Taillard, E. 2010. Scheduling Instances. Available via <http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html> [accessed April 14, 2010].

Widmer, M., and A. Hertz. 1989. A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research* 41(2):186-193.

Zobolas, C., C. Tarantilis, and G. Ioannou. 2009. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research* 36:1249-1267.

AUTHOR BIOGRAPHIES

ANGEL A. JUAN is an Associate Professor of Simulation and Data Analysis in the Computer Science Department at the Open University of Catalonia, Spain. He is also a Lecturer at the Technical University of Catalonia, Spain. He holds a Ph.D. in Industrial Engineering, an M.S. in Information Technologies, and a M.S. in Applied Mathematics. His research interests include computer simulation, educational data analysis and mathematical e-learning. He is an editorial board member of the *Int. J. of Data Analysis Techniques and Strategies* and the *Int. J. of Information Systems & Social Change*. He is also a member of the INFORMS society. His webpage and e-mail address are <http://ajuanp.wordpress.com> and <ajuanp@gmail.com>.

RUBEN RUIZ is an Associate Professor of Heuristics and Algorithms in the Dep. of Statistics and Operations Research at the Universidad Politécnica de Valencia, Spain. He holds a Ph.D. and an M.S. in Computer Science. His research interests include combinatorial optimization and its applications to real-life scenarios. He is editor of the *European Journal of Industrial Engineering*. His webpage and e-mail address are <http://soa.iti.es/rruiz> and <rruiz@eio.upv.es>.

HELENA R. LOURENÇO is an Associate Professor of Operations Research and Logistics at the Department of Economics and Business at the Universitat Pompeu Fabra, Spain. She holds a Ph.D. and an M.S. in Operations Research. Her research interests include Operations Research, Operations Management, Logistics, Metaheuristics, Combinatorial Optimization and Scheduling. Her webpage and e-mail address are, respectively: <http://www.econ.upf.edu/~ramalhin/welcome.htm> and <helena.ramalhinho@upf.edu>.

MANUEL MATEO is an Associate Professor in the Department of Management at Universitat Politécnica de Catalunya (UPC), Spain. He holds a Ph.D. and a M.S. in Industrial Engineering. His courses are related to Industrial Engineering and some of them describe problems of Operations Research. His research field is focused on scheduling: hoist scheduling problem, flow-shop, problems in the supply chain, etc. He is the author or co-author of some papers and brochures. He is Lecturer at the Universitat Oberta de Catalunya (UOC), as well as a member of ADINGOR and SEIO societies. His e-mail address is <manel.mateo@upc.edu>.

DRAGOS IONESCU is a student of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. At MIT, he has hold lab assistant positions for courses related to software engineering and transmission systems. Recently, he has been working as a junior research assistant at the Open University of Catalonia (Spain), as part of the IN3-HAROSA Knowledge Community. His e-mail address is <dionescu@mit.edu>.