Hybrid Evolutionary Algorithm for Flowtime Minimisation in No-Wait Flowshop Scheduling

Geraldo Ribeiro Filho[1], Marcelo Seido Nagano[2], and Luiz Antonio Nogueira Lorena[3]

[1] Faculdade Bandeirantes de Educação Superior
R. José Correia Gonçalves, 57
08675-130, Suzano - SP - Brazil
geraldorf@uol.com.br
[2] Escola de Engenharia de São Carlos - USP
Av. Trabalhador São-Carlense, 400
13566-590, São Carlos - SP - Brazil
drnagano@usp.br
[3] Instituto Nacional de Pesquisas Espaciais - INPE/LAC
Av. dos Astronautas, 1758
12227-010, São José dos Campos - SP - Brazil
lorena@lac.inpe.br

Abstract. This research presents a novel approach to solve m-machine no-wait flowshop scheduling problem. A continuous flowshop problem with total flowtime as criterion is considered applying a hybrid evolutionary algorithm. The performance of the proposed method is evaluated and the results are compared with the best known in the literature. Experimental tests show the superiority of the evolutionary hybrid regarding the solution quality.

1 Introduction

This paper considers the m-machine no-wait flowshop scheduling problem. In a no-wait flowshop, the operation of each job has to be processed without interruptions between consecutive machines, i.e., when necessary, the start of a job on a given machine must be delayed so that the completion of the operation coincides with the beginning of the operation on the following machine. Applications of no-wait flowshops can be found in many industries. For example, in steel factories, the heated metal continuously goes through a sequence of operations before it is allowed to cool in order to prevent defects in the composition of the steel. A second example is a plastic product that requires a series of processes to immediately follow one another in order to prevent degradation. Similar situations arise in other process industries such as the chemical and pharmaceutical. Hall and Sriskandarajah [1] give in their survey paper a detailed presentation of the applications and research on this problem and indicate that the problem with the objective of total or mean completion time is NP-Complete in the strong

sense even for the two-machine case. Considering that flowtime or completion time of a job is the same when the job is ready for processing at time zero and that minimizing total or mean completion time are equivalent criteria, some of the works on the no-wait problem with the objective of minimizing any of these criteria include Adiri and Pohoryles [2], Rajendran and Chaudhuri [3], Van der Veen and Van Dal [4], Chen et al. [5], Aldowaisan and Allahverdi [6], Aldowaisan [7], and Allahverdi and Aldowaisan [8].

Chen et al. [5] later develop a genetic algorithm and compare it with the heuristics of Rajendran and Chaudhuri [3]. Aldowaisan and Allahverdi [9] proposed six heuristics for the no-wait flowshop with the objective of minimizing the total completion time, which perform better than the heuristics of Rajendran and Chaudhuri [3] and Chen et al. [5]. Aldowaisan and Allahverdi [10] proposed simulated annealing (SA) and GA-based heuristics for the no-wait flowshop scheduling problem with the makespan criterion by incorporating a modified Nawaz-Enscore-Ham (NEH) heuristic (see Nawaz et al. [11]), based on a new insertion technique and pairwise procedure.

Fink and Voβ [12] presented some construction methods and metaheuristics for the no-wait flowshop scheduling problem with the total flowtime criterion. Construction methods presented were the NN, the Chins, and the Pilot heuristics whereas metaheuristics investigated were steepest descent (SD), iterated steepest descent (ISD), SA, and TS algorithms. See Fink and Voβ [12] for the details of construction methods and metaheuristics. The application of above heuristics for the no-wait flowshop scheduling problem were based on HotFrame, a heuristic optimization tool developed by Fink and Voβ [13]. All heuristics have been applied to the benchmark suite of Taillard [14], originally generated for the unrestricted permutation flowshop sequencing problem. In addition, Fink and Voβ [12] provided a detailed analysis of construction methods, different neighborhood structures embedded in SD, ISD, SA and TS algorithms. According to results, SA and reactive tabu search (RTS) algorithms generated better results with a 1000s CPU time in connection with shift (insert) neighborhood on the basis of initial solutions provided by Chins and Pilot-10 heuristics.

Recently Pan Q-K., et al. [15] presented a discrete particle swarm optimization (DPSO) to solve the no-wait flowshop scheduling problem with both makespan and total flowtime criteria. The main contribution of this study is due to the fact that particles are represented as discrete job permutations and a new position update method is developed based on the discrete domain. In addition, the DPSO algorithm is hybridized with the variable neighborhood descent (VND) algorithm to further improve the solution quality. Several speed-up methods are proposed for both the swap and insert neighborhood structures. The DPSO algorithm is applied to both 110 benchmark instances of Taillard [14] by treating them as the no-wait flowshop problem instances with the total flowtime criterion, and to 31 benchmark instances provided by Carlier [16], Heller [17], and Reeves [18] for the makespan criterion. For the makespan criterion, the solution quality is evaluated according to the reference makespans generated by Rajendran [19] whereas for the total flowtime criterion, it is evaluated with the

optimal solutions, lower bounds and best known solutions provided by Fink and Voβ [12]. The computational results show that the DPSO algorithm generated either competitive or better results than those reported in the literature. Ultimately, 74 out of 80 best known solutions provided by Fink and Voβ [12] were improved by the VND version of the DPSO algorithm. Based on the literature examination we have made, the aforementioned metaheuristic presented by Pan Q-K., et al. [15] yields the best solutions for total flowtime minimization in a permutation no-wait flowshop. In this paper, we address the generic m-machine no-wait flowshop problem with the objective of minimizing total flowtime. We propose a new Hybrid Evolutionary metaheuristic Algorithm and compare with the heuristic of Pan Q-K., et al. [15].

2 Evolutionay Heuristic

We have used in this work a Evolutionary Heuristics (EH) based on classic Genetic Algorithms. The chromosome representation used in EH was a n positions array, were each position indicates a task in the solution schedule. The population size was fixed in 200 individuals empirically, to make room for good individuals in the initial population altogether with randomly generated individuals to give some degree of diversification.

As the quality of the individual in the initial population has great importance in the evolutionary strategies, we have tried to create such quality individuals with a variation of the heuristic called NEH presented by Nawaz et al. [11]. The original form of NEH initially sorts a set of n tasks according to non-descending values of the sum of task processing times by all machines. The two first tasks in the sorted sequence are scheduled to minimize the partial flow time. The remaining tasks are then sequentially inserted into the schedule in the position that minimizes the partial flow time. In the variation used in this work, after the sort, the first two tasks to be scheduled were randomly taken. The very first individual inserted into the population was generated by NEH, the variation of NEH was used to generate other individuals in a number given by

$$\min\left(\frac{n*(n-1)}{4}, \frac{200}{2}\right) \quad , \tag{1}$$

and the remain part of the initial population was filled with randomly generated schedules.

The evaluation of the individuals was made by the minimization of the total flow time for no-wait flowshop. The individual insertion routine kept the population sorted, and the best individual, the one with the lowest total flow time, occupied the first position in the population. The insertion routine was also responsible for maintain only one copy of each individual in the population.

Twenty five new individuals were created by iteration, and possibly inserted into the population. The stop condition used was the maximum of 200 iterations or 20 consecutive iterations with no new individuals being inserted. All these parameters had its values chosen empirically as result of tests.

A new individual generation was made by randomly selecting two parents, one from the best 20% of the population, called the base parent, and the other from the entire population, called the guide parent. A crossover process known as Block Order Crossover (BOX), presented by Syswerda [20], was applied to both parents, generating a single offspring by copying blocks of genes from the parents. In this work the offspring was generated with 70% of its genes coming from the base parent. Several other recombination operators are studied and empirically evaluated by Cotta and Troya [21]. Investigation regarding position-oriented recombination operators is also possible in further studies.

After the crossover, the offspring had a probability of 70% to be improved by a local search procedure called LS1, shown in Figure 1. This procedure used two neighborhood types: permutation and insertion. The permutation neighborhood around an individual was obtained by swapping every possible pair of chromosome genes, producing $n * (n-1)/2$ different individuals. The insertion neighborhood was obtained by removing every gene from its position, and inserting it in each other position in the chromosome, producing $n * (n-1)$ different individuals. Both, the improvement probability and the number of genes coming from the base parent to the offspring were parameters which values were taken after several tests.

```
Procedure LS1(current_solution)
Begin
  cs = current_solution;
  stop = false;
  While (stop == false) do Begin
    P = Permutation_neighborhood(cs);
    sp = First s in P that eval(s) < eval(cs), or eval(s) < eval(t) for all t in P;
    I = Insertion_neighborhood(cs);
    si = First s in I that eval(s) < eval(cs), or eval(s) < eval(t) for all t in I;
    If (eval(sp) < min(eval(si), eval(cs))) then
      cs = sp;
    else
      If (eval(si) < min(eval(sp), eval(cs))) then
        cs = si;
      else
        stop = true;
  End;
  Return cs;
End
```

Fig. 1. Pseudo-code for the LS1 Local Search Procedure

The new individual was then inserted into the population in the position relative to its evaluation, shifting ahead the subsequent part of the population, and therefore removing the last, and worst, individual.

At the end of the EH iterations, the very first individual in the population was considered as the best solution found so far.

To evaluate the EH, tests were made using the first Taillard [14] instances considered as no-wait flowshop, with n=20 tasks and m=5, 10 and 20 machines. The code was written in C language and ran in a Pentium IV, 3.0 GHz, 1Gb RAM computer. The EH ran 10 times for each instance and the EH has obtained exactly the same as best solution values found so far, compared with the work of Fink and Voβ [12] and Pan et al. [15].

The preliminary tests with the Taillard next instances, with 50 and 100 tasks, were showing not so promising results. Therefore, we have adapted the EH to a hybrid form using the Cluster Search (CS) algorithm, following described.

3 Clustering Search

The metaheuristic Clustering Search (CS), proposed by Oliveira and Lorena [22, 23], consists of a solution clustering process to detect supposedly promising regions in the search space. The objective of the detection of these regions as soon as possible is to adapt the search strategy. A region can be seen as a search subspace defined by a neighborhood relation.

The CS has an iterative clustering process, simultaneously executed with a heuristic, and tries to identify solution clusters that deserve special interest. The regions defined by these clusters must be explored, as soon as they are detected, by problem specific local search procedures. The expected result of more rational use of local search is convergence improvement associated with reduction of computational effort.

CS tries to locate promising regions by using clusters to represent these regions. A cluster is defined by a triple $G = (C, r, \beta)$ where C, r and β are, respectively, the center, the radius of a search region around the center, and a search strategy associated with the cluster.

The center C is a solution that represents the cluster, identify its location in the search space, and can be changed along the iterative process. Initially the centers can be obtained randomly, and progressively tend to move to more promising points in the search space. The radius r defines the maximum distance from the center to consider a solution being inside the cluster. For example, the radius r could be defined as the number moves to change a solution into another. The CS admits a solution to be inside of more than one cluster. The strategy β is a procedure to intensify the search, in which existing solutions interact with each other to create new ones.

The CS consists of four components, conceptually independent, with different attributions: a metaheuristic (ME), an iterative clustering process (IC), a cluster analyzer (CA), and a local optimization algorithm (LO).

The ME component works as a full time iterative solution generator. The algorithm is independently executed from the other CS components, and must be able to continuously generate solutions for the clustering process. Simultaneously, the clusters are maintained as containers for these solutions. This process works as a loop in which solutions are generated along the iterations.

The objective of the IC component is to associate similar solutions to form a cluster, keeping a representative one of them as the cluster center. The IC is implemented as an online process where the clusters are feed with the solutions produced by the ME. A maximum number of clusters is previously defined to avoid unlimited cluster generation. A distance metric must be defined also previously to evaluate solutions similarity for the clustering process. Each solution received by IC is inserted into the cluster having the center most similar to it, causing a perturbation in this center. This perturbation is called assimilation and consists of the center update according to the inserted solution.

The CA component provides an analysis of each cluster, at regular time intervals, indicating probable promising clusters. The so called cluster density λ_i measures the i-th cluster activity. For simplicity, λ_i can be the cluster's number of assimilated solutions. When λ_i reach some threshold, meaning that ME has produced a predominant information model, the cluster must be more intensively investigated to accelerate its convergence to better search space regions. CA is also responsible for the removal of low density clusters, allowing new and better clusters to be created, while preserving the most active clusters. The clusters removal does not interfere with the set of solutions being produced by ME, as they are kept in a separate structure.

Finally, the LO component is a local search module that provides more intensive exploration of promising regions represented by active clusters. This process runs after CA has determined a highly active cluster. The local search corresponds to the β element that defines the cluster and is a problem specific local search procedure.

4 Evolutionary Clustering Search for the No-Wait Flow Shop Problem

This research has used a metaheuristic called Evolutionary Clustering Search (ECS) proposed by Oliveira and Lorena [22, 23] that combines Evolutionary Heuristics (EH) and Clustering Search, and has originally applied it to the No-Wait Permutation Flow Shop problem. The ECS uses an EH to implement the ME component of the CS and generate solutions that allow the exploration of promising regions. A pseudo-code representation of the ECS for the No-Wait Flowshop problem is shown in Figure 2.

As ECS has presented good performance in previous applications, and considering the accelerated convergence provided by CS when compared with pure, non hybrid, algorithms, the aim of this work was to attempt to beat the best results recently produced and found in the literature, even with larger computer times, characteristic of evolutionary processes. Seeking originality, this was another reason to apply CS in this research.

The Evolutionary Clustering Search for No-Wait Flow Shop (ECS-NWFS) presented in this work has used the EH presented in Section 2 with some different parameters values. The population was fixed in 500 individuals to make room for more individuals generated by the NEH variation with larger number of tasks.

```
Procedure ECS-NWFS()
Begin
  Initialize population P;
  Initialize clusters set C;
  While (stop condition == false) do Begin
    While (i < new_individuals) do Begin
      parent1 = Selected from best 40% of P;
      parent2 = Selected from the whole P;
      offspring = Crossover(parent1, parent2);
      Local_Search_LS1(offspring) with 60% probability;
      If (Insert_into_P(offspring))
        Assimilate_or_create_cluster(offspring, C);
      i = i + 1;
    End;
    For each cluster c in C
      If (High_assimilation(c))
        Local_Search_LS2(c);
  End;
End;
```

Fig. 2. Pseudo-code for the ECS algorithm

Therefore, the number of such individuals was given by Equation 1 adapted to the new population size. At each iteration, 50 new individuals were generated and possibly inserted into the population. The stop condition used was a maximum of 500 iterations or 20 consecutive iterations with no insertions. New individuals generation was made the same way, using base and guide parents with BOX crossover. All other parameter were kept the same and these new values were obtained empirically with several tests.

A cluster set initialization process was created to take advantage of the good individuals in the EH initial population. This routine scanned the population, from the best individual to the worst, creating new clusters or assimilating the individuals into clusters already created. A new cluster was created when the distance from the individual to the center of any cluster was larger than $r = 0.85 * n$, and the individual was used to represent the center of the new cluster. Otherwise, the individual was assimilated by the cluster with the closest center. The distance measure from an individual to the cluster center was taken as the number of swaps necessary to transform the individual into the cluster center. Starting from the very first, each element of the individual was compared to its equivalent in the cluster center, at the same position. When non coincident elements were found the rest of the individual chromosome was scanned to find the same element found in the cluster center, and make a swap. At the end, the individual was identical to the cluster center, and the number of swaps was considered as a distance measure. The clusters initialization process ended when the whole population was scanned or when a maximum of 450 clusters were created. The cluster radius and the maximum number of clusters were chosen after several tests.

The assimilation of an individual by a cluster was based on the Path Relinking procedure presented by Glover [24]. Starting from the individual chromosome,

successive swaps were made until the chromosome became identical to the cluster center. The pair of genes chosen to swap was the one that more reduced, or less increased, the chromosome total flow time. At each swap the new chromosome configuration was evaluated. At the end of the transformation, the cluster center was moved to (replaced by) the individual, or the intermediary chromosome, that has the best evaluation better than the current center. If no such improvement was possible, the cluster center remains the same.

The successfully inserted individuals were then processed by the IC component of ECS-NWFS. This procedure tried to find the cluster having the closest center and of which radius r the individual was within. When such cluster could be found, the individual was assimilated, otherwise, a new cluster was created having the individual as its center. New clusters were created only if the ECS-NWFS had not reached the clusters limit. Tests have shown that the number of cluster tends to increase at very first iterations, and slowly decrease as iterations continue and the ECS removes the less active clusters.

After the generation of each new individual by the EH, its improvement by LS1, and the insertion into the population, the ECS-NWFS executed its cluster analysis procedure, with two tasks: remove the clusters that had no assimilations in the last 5 iterations, and take every cluster that had any assimilation in the current iteration and ran it through a second local optimization procedure, called LS2 and shown in Figure 3.

```
Procedure LS2(current_solution)
Begin
  cs = current_solution;
  stop = false;
  While (stop == false) do Begin
    I = Insertion_neighborhood(cs);
    si = First s in I that eval(s) < eval(cs), or eval(s) < eval(t) for all t in I;
    If (eval(si) < eval(cs)) then Begin
      cs = si;
      P = Permutation_neighborhood(cs);
      sp = First s in P that eval(s) < eval(cs), or eval(s) < eval(t) for all t in P;
      If (eval(sp) < eval(cs)) then
        cs = sp;
    End else Begin
      Pnh = Permutation_neighborhood(cs);
      sp = Scan Pnh until sp is better than cs, or sp is the best in Pnh;
      If (eval(sp) < eval(cs))
        cs = sp;
      else
        stop = true;
    End;
  End;
  Return cs;
End
```

Fig. 3. Pseudo-code for the LS2 Local Search Procedure

Along the ECS-NWFS processing the best cluster was kept saved. At the end of the execution, the center of the best cluster found so far was taken as the final solution produced by the algorithm.

5 Computational Experiments with ECS-NWFS

The ECS-NWFS ran 10 times for each instance and Table 1 shows that, except by one single instance, the algorithm has obtained better results than the best found by Fink and Voβ [12] and Pan et al. [15] for the Taillard instances with n=50 and 100 tasks. The table also shows the average of 10 runs of the ECS-NWFS, and the percentage gap between this average and the best solution. The gap was very low, less than 0.5% for all test instances.

Table 1. New Best Known solution for Taillard's benchmarks with n=50 tasks and m=5 machines (Ta031-Ta040), m=10 (Ta041-Ta050), m=20 (Ta051-Ta060), and n=100 tasks with m=5 machines (Ta061-Ta070), m=10 (Ta071-Ta080) and m=20 (Ta081-Ta090), considered as No-Wait Permutation Flow Shop, for Flowtime minimisation

Inst	*F&V*	*DPSO*	*ECS*	Avg	%Gap	Inst	*F&V*	*DPSO*	*ECS*	Avg	%Gap
Ta031	76016	75682	**75668**	75674.2	0.01	Ta061	308052	303750	**303567**	304736.3	0.39
Ta032	83403	82874	**82874**	83028.2	0.19	Ta062	302386	297672	**296321**	297472.3	0.39
Ta033	78282	78103	**78103**	78112.3	0.01	Ta063	295239	291782	**290638**	291406.2	0.26
Ta034	82737	82467	**82359**	82370.6	0.01	Ta064	278811	277093	**274722**	275266.1	0.20
Ta035	83901	83493	**83476**	83476.0	0.00	Ta065	292757	289554	**288344**	288766.4	0.15
Ta036	80924	80749	**80671**	80685.1	0.02	Ta066	290819	287055	**285752**	286502.3	0.26
Ta037	78791	78604	**78604**	78628.8	0.03	Ta067	300068	297731	**296537**	297383.7	0.29
Ta038	79007	78796	**78672**	78707.7	0.05	Ta068	291859	287754	**285961**	286890.1	0.32
Ta039	75842	75825	**75639**	75709.1	0.09	Ta069	307650	304131	**303657**	304119.4	0.15
Ta040	83829	83430	**83430**	83507.0	0.09	Ta070	301942	298119	**296964**	297683.6	0.24
Ta041	114398	114051	**113908**	113984.4	0.07	Ta071	412700	409715	**407679**	408079.3	0.10
Ta042	112725	112427	**112180**	112255.0	0.07	Ta072	394562	390417	**389001**	389884.5	0.23
Ta043	105433	105345	**105345**	105357.1	0.01	Ta073	405878	402274	**402036**	402483.7	0.11
Ta044	113540	113367	**113201**	113273.2	0.06	Ta074	422301	417733	**417091**	417545.9	0.11
Ta045	115441	115295	**115295**	115335.8	0.04	Ta075	400175	397049	**395519**	396701.0	0.30
Ta046	112645	112459	**112459**	112481.2	0.02	Ta076	391359	387398	**386418**	387696.8	0.33
Ta047	116560	116631	**116444**	116453.7	0.01	Ta077	394179	391057	**390076**	390798.6	0.19
Ta048	115056	115065	**114945**	114973.5	0.02	Ta078	402025	399214	**397072**	398109.4	0.26
Ta049	110482	110367	**110367**	110371.8	0.00	Ta079	416833	413701	**411396**	412792.1	0.34
Ta050	113462	113427	**113427**	113427.0	0.00	Ta080	410372	406206	**406001**	406311.1	0.08
Ta051	172845	172981	**172740**	172774.1	0.02	Ta081	562150	558199	**556564**	557322.9	0.14
Ta052	161092	160836	**160739**	160745.5	0.00	Ta082	563923	561305	**559171**	560357.8	0.21
Ta053	160213	160104	**160104**	160263.0	0.10	Ta083	562404	560530	**558440**	559562.1	0.20
Ta054	161557	161690	**161492**	161549.0	0.04	Ta084	562918	559690	**557386**	558275.3	0.16
Ta055	167640	167336	**167081**	167081.0	0.00	Ta085	556311	551388	**550704**	551675.1	0.18
Ta056	161784	161784	**161460**	161558.9	0.06	Ta086	562253	558356	**557051**	558196.7	0.21
Ta057	167233	**167064**	167098	167099.7	0.02	Ta087	574102	571680	**568667**	569486.6	0.14
Ta058	168100	167822	**167822**	168020.9	0.12	Ta088	578119	574269	**572945**	574158.1	0.21
Ta059	165292	165207	**165207**	165215.5	0.01	Ta089	564803	560710	**557946**	559497.0	0.28
Ta060	168386	168386	**168386**	168386.0	0.00	Ta090	572798	568927	**566054**	567688.8	0.29

The quality of the initial population individuals, allied to diversity, and the performance of the local search routines, can be considered key factors for the quality of the final solutions.

Processing times had a large variation from 48 seconds for an instance with 50 tasks, to 1 hour and 3 seconds for an instance with 100 tasks.

6 Conclusion

The main objective of this work was apply CS to the No-Wait Permutation Flow Shop Scheduling Problem of original and inedited form. Experimental results presented in the tables have shown that the EH had comparable, and the ECS-NWFS method had superior performance compared with the best results found in the literature for the considered test problems, using the in-process inventory reduction, or minimization of the total flow time, as the performance measure. The computational effort was acceptable for practical applications.

The classic optimization problem of task schedule in No-Wait Flow Shop has been the object of intense research for decades. For practical applications this problem may be considered already solved, although, because of its complexity it still remains as a target for the search for heuristic and metaheuristic methods.

The research related in this paper was motivated by the above considerations, and have tried to rescue the essential characteristics of metaheuristic methods, balance between solution quality and computational efficiency, simplicity and implementation easiness.

References

1. Hall, N.G., Sriskandarajah, C.: A survey of machine scheduling problems with blocking and no-wait in process. Operations Research, **44** (1996) 510–525
2. Adiri I., Pohoryles D.: Flowshop/no-idle or no-wait scheduling to minimize the sum of completion times. Naval Research Logistics Quarterly. **29** (1982) 495–504
3. Rajendran C., Chaudhuri D.: Heuristic algorithms for continuous flow-shop problem. Naval Research Logistics. **37** (1990) 695–705
4. Van der Veen, J.A.A., Van Dal, R.: Solvable cases of the no-wait flowshop scheduling problem. Journal of the Operational Research Society. **42** (1991) 971–980
5. Chen C.L., Neppalli R.V., Aljaber N.: Genetic algorithms applied to the continuous flow shop problem. Computers and Industrial Engineering. **30** (1996) 919–929
6. Aldowaisan T., Allahverdi A.: Total flowtime in no-wait flowshops with separated setup times. Computers and Operations Research. **25** (1998) 757–65
7. Aldowaisan T.: A new heuristic and dominance relations for no-wait flowshops with setups. Computers and Operations Research, **28** (2000) 563–584
8. Allahverdi A., Aldowaisan T.: No-wait and separate setup three-machine flowshop with total completion time criterion. International Transactions in Operational Research. **7** (2000) 245–264
9. Aldowaisan T., Allahverdi A.: New heuristics for m-machine no-wait flowshop to minimize total completion time. Omega. **32(5)** (2004) 345–352
10. Aldowaisan T., Allahverdi A.: New heuristics for no-wait flowshops to minimize makespan. Computers and Operations Research. **30(8)** (2003) 1219–1231
11. Nawaz, M., Enscore, E. E. and Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA, **11** (1983) 91–95

12. Fink A., Voβ S.: Solving the continuous flow-shop scheduling problem by metaheuristics. European Journal of Operational Research. **151** (2003) 400-414
13. Fink A., Voβ S.: HotFrame: a heuristic optimization framework. In: Vo S., Woodruff D., editors. Optimization software class libraries. Kluwer, Boston (2002) 81–154
14. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research. **64** (1993) 278–285
15. Pan Q.K., Tasgetiren M.F., Liang Y.C.: A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. Computers and Operations Research (2007) doi: 10.1016/j.cor.2006.12.030.
16. Carlier J.: Ordonnancements a contraintes disjonctives. RAIRO Recherche operationelle. **12** (1978) 333-351
17. Heller J. Some numerical experiments for an MxJ flow shop and its decision-theoretical aspects. Operations Research. **8** (1960) 178–184
18. Reeves C.: A genetic algorithm for flowshop sequencing. Computers and Operations Research. 22 (1995) 5–13
19. Rajendran C.: A no-wait flowshop scheduling heuristic to minimize makespan. Journal of the Operational Research Society. **45** (1994) 472-479
20. Syswerda, G.: Uniform crossover in genetic algorithms. In: International Conference on Genetic Algorithms (ICGA). Virginia, USA, (1989) 2–9
21. Cotta, C., Troya, J. M.: Genetic Forma Recombination in Permutation Flowshop Problems. Evolutionary Computation. **6** (1998) 25–44
22. Oliveira, A. C. M. and Lorena, L. A. N.: Detecting promising areas by evolutionary clustering search. In: Advances in Artificial Intelligence, Bazzan, A.L.C. and Labidi, S. (eds) Springer Lecture Notes in Artificial Intelligence. (2004) 385–394
23. Oliveira, A. C. M. and Lorena, L. A. N.: Hybrid evolutionary algorithms and clustering search. In: Crina Grosan,Ajith Abraham and Hisao Ishibuchi (eds) Hybrid Evolutionary Systems - Studies in Computational Intelligence - Springer SCI Series. **75** (2007) 81–102
24. Glover, F.: Tabu search and adaptive memory programing: Advances, applications and challenges. In: Interfaces in Computer Science and Operations Research. Kluwer. (1996) 1–75