Evolutionary Clustering Search for Flowtime Minimization in Permutation Flow Shop

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 paper deals with the Permutation Flow Shop scheduling problem with the objective of minimizing total flow time, and therefore reducing in-process inventory. A new hybrid metaheuristic Genetic Algorithm - Cluster Search is proposed for the scheduling problem solution. The performance of the proposed method is evaluated and results are compared with the best reported in the literature. Experimental tests show the new method superiority for the test problems set, regarding the solution quality.

1 Introduction

This paper deals with Permutation Flow Shop scheduling problems, which consists of finding a sequence for the jobs that optimises some schedule performance measure. Usually, such measures are the maximum completion time (makespan), and the total flowtime. As it is well known, the first measure is associated with an efficient utilization of resources, and the second one with a faster response to job processing, therefore reducing in-process inventory. In this paper we introduce a hybrid meta-heuristic method with the objective of minimizing the total flowtime.

This production scheduling problem is NP-complete [9, 26], therefore the search for an optimal solution is of more theoretical than practical importance.

In the last ten years a number of heuristic methods have been introduced with the objective of minimizing total flowtime, or equivalently the mean flowtime in permutation flow shops. These heuristic methods can be divided into two main classes: construction methods and improvement methods. The literature on construction methods includes the heuristics proposed by Ahmadi and Bagchi [1], Rajendran and Chaudhuri [22], Rajendran [23], Ho [12], Wang et al. [31], Woo and Yim [33], Liu and Reeves [15], Allahverdi and Aldowaisan [2], Framinan and Leisten [8], Framinan et al. [7], Li et al. [14] and Nagano and Moccellin [18].

Improvement methods such as ant-colony optimization algorithm proposed by Rajendran and Ziegler [25] and swarm optimization algorithm proposed by Tasgetiren et al. [30], start from an initial permutation, which is usually generated by a construction method, and then iteratively generate a sequence of improved permutations. It is obvious that improvement methods generate significantly better solutions than construction ones.

Rajendran and Ziegler [25] introduce two heuristics. The first algorithm extends the ideas of the ant-colony algorithm by Stuetzle [28], called max-min ant system (MMAS), by incorporating the summation rule suggested by Merkle and Middendorf [16] and a new proposed local search technique. The second ant-colony algorithm is newly developed. These ant-colony algorithms were applied to 90 benchmark problems taken from Taillard [29]. Considering the minimization of makespan the comparison shows that the two proposed ant-colony algorithms perform better, on an average, than the MMAS. Subsequently, by considering the objective of minimizing the total flowtime of jobs, a comparison of solutions yielded by the proposed ant-colony algorithms with the best heuristic solutions known for the benchmark problems, as published in an extensive study by Liu and Reeves [15], is carried out. The comparison shows that the proposed ant-colony algorithms are clearly superior to the heuristics analyzed by Liu and Reeves. For 83 out of 90 problems considered, better solutions have been found by the two proposed ant-colony algorithms, as compared to the solutions reported by Liu and Reeves [15].

Like many optimization problems, scheduling are commonly approached by evolutionary techniques. Cotta and Fernandez [6] applied memetic algorithms to planning, scheduling and timetabling, and Kleeman and Lamont [13] have studied multi-objective evolutionary algorithms (MOEA) with fixed and variable chromosome length applied to the flow-shop and job-shop problems.

Recently Tasgetiren et al. [30] presented a particle swarm optimization algorithm (PSO) to solve the permutation flow shop sequencing problem with the objectives of minimizing makespan and the total flowtime of jobs. For this purpose, a heuristic rule called the smallest position value (SPV) borrowed from the random key representation of Bean [3] was developed to enable the continuous particle swarm optimization algorithm to be applied to all classes of sequencing problems. In addition, a very efficient local search, called variable neighborhood search (VNS), was embedded in the PSO algorithm to solve the well known benchmark suites in the literature. The PSO algorithm was applied to both the 90 benchmark instances provided by Taillard [29], and the 14,000 random, narrow random and structured benchmark instances provided by Watson et al. [32]. For makespan criterion, the solution quality was evaluated according to the best known solutions provided either by Taillard [29], or Watson et al. [32]. The total flowtime criterion was evaluated with the best known solutions provided by Liu and Reeves [15], and Rajendran and Ziegler [25]. For the total flowtime criterion, 57 out of the 90 best known solutions reported by Liu and Reeves [15], and Rajendran and Ziegler [25] were improved whereas for the makespan criterion, 195 out of the 800 best known solutions for the random and narrow random problems reported by Watson et al. [32] were improved by the VNS version of the PSO algorithm.

Based on the literature examination we have made, the aforementioned metaheuristic PSO-VNS presented by Tasgetiren et al. [30] yields the best solutions for total flowtime minimization in a permutation flow shop.

2 Clustering Search

The metaheuristic Clustering Search (CS), proposed by Oliveira and Lorena [20, 21], 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). Figure 1 shows a representation of the four components, the search space and the clusters centers.

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.

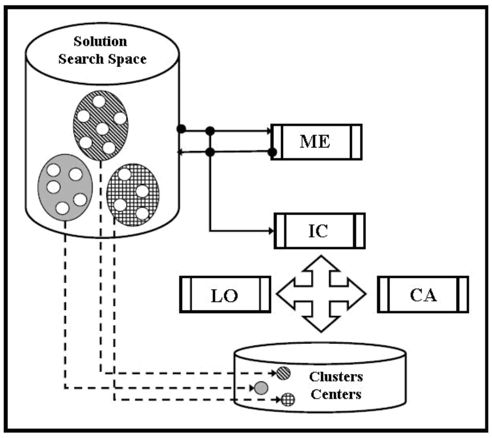

Fig. 1. Clustering Search Conceptual Diagram

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.

3 Evolutionary Clustering for the Permutation Flow Shop Problem

This research has used a metaheuristic called Evolutionary Clustering Search (ECS) proposed by Oliveira and Lorena [20, 21] that combines Genetic Algorithms (GA) and Custering Search, and has applied it to the Permutation Flow Shop problem. The ECS uses a GA to implement the ME component of the CS and generate solutions that allow the exploration of promising regions by the other components of CS. A pseudo-code representation of the ECS is shown in Figure 2.

```
Procedure ECS-FS()
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

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 Flow Shop (ECS-FS) presented in this work has some modifications from the original CS general concept presented in the previous section.

As the quality of the individuals in the initial population is important for the GA performance, to ensure this quality, the population initialization was done with a variation of the method known as NEH, presented by Nawaz et al. [19]. The original form of NEH initially sort 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.

The chromosome representation used in the GA was a n element vector, one element for each task, storing the position of that task in the solution schedule. After several tests, the population size was fixed in 500 individuals to make room for good individuals produced by NEH and its variation, together with randomly generated individuals to provide diversity.

The very fist individual inserted into the initial population was generated by the NEH procedure. Part of the other individuals was generated by a variation of the NEH in which the two tasks from the sorted sequence to be first scheduled were randomly chosen from the whole sequence. The rest of the sequence was then scheduled the same way as the original NEH.

To ensure some degree of diversity in the initial population, the maximum number of individuals generated by the modified NEH was given by

$$\min\left(\frac{n*(n-1)}{4}, \frac{500}{2}\right) \quad , \tag{1}$$

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

The evaluation of the population individuals was made by the minimization of the total flow time. 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.

A cluster set initialization process was created to take advantage of the good individuals in the GA 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 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 200 clusters were created. Both the cluster radius and the maximum number of clusters are parameters which values were chosen after several tests, with the objective to work with all problem classes used for tests.

The assimilation of an individual by a cluster was based on the Path Relinking procedure presented by Glover [10]. 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.

At each iteration of the GA, 50 new individuals were created and possibly inserted into the population. The stop condition used was the maximum of 100 iterations or 20 consecutive iterations with no new individuals being inserted, as the population could have one single copy of each individual.

The new individual generation was made by randomly selecting two parents, one from the best 40% 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 Crossove (BOX), presented by Syswerda [27], was then applied to both parents, generating a single offspring by copying blocks of genes from the parents. In this work the offspring was generated with 50% of its genes coming from each parent. Several other recombination operators are studied and empirically evaluated by Cotta and Troya [5]. Investigation regarding position-oriented recombination operators are also possible in further studies. Figure 3 illustrates the BOX crossover.

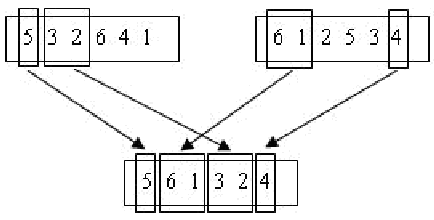

Fig. 3. BOX Crossover

The number of new individuals created at each iteration, the stop condition, the part of the population from which comes the base parent for crossover, and contribution of each parent in the crossover process are all parameters which values were obtained after several tests.

After the crossover, the offspring had a probability of 60% to be improved by a local search procedure called LS1, shown in Figure 4.

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.

```
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. 4. 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.

The successfully inserted individuals were then processed by the IC component of ECS-FS. This procedure tried to find the cluster having the closest center and of which radius r the individual was within. When such cluster could 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-FS had not reached the 200 clusters limit. Tests have shown that the number of cluster tends to increase at very first ECS iterations, and slowly decrease as iterations continue and the ECS removes the less active clusters.

After the generation of new individuals, its improvement and insertion into the population, the ECS-FS executed its CA component. This cluster analysis procedure performed 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 local optimization procedure, called LS2 and shown in Figure 5, corresponding to the LO component of ECS-FS.

Again, the probability with which an offspring ran trough local search before being inserted into the population and the number of iterations without assimilation used to delete clusters was parameters which values were chosen after several tests.

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

```
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. 5. Pseudo-code for the LS2 Local Search Procedure

4 Computational Experiments

The performance evaluation of the proposed hybrid heuristic method ECS-FS, was made through computational experiments using the Taillard [29] test problems. These problem are divided into n tasks and m machines sets, each set having ten instances. Results were compared with those reported in the works of Liu and Reeves [15], Rajendran and Ziegler [25], Li et al. [14] and Tasgetiren et al. [30].

For this work, the ECS-FS code was written in the C programming language and was executed on a Pentium IV, 3.0 GHz, 1 GByte RAM personal computer.

Two statistic measures were used to performance evaluation: the success rate and the average relative deviation. The first is given by the ratio between the number of problems for which a method produced the minimum total flow time given by all compared methods, and the number of problems solved in a problem set. The second shows the deviation obtained by a method h from the minimum total flow time as above, and is given by

$$RD_h = \left(\frac{F_h - F_*}{F_*} \right) \tag{2}$$

where F_h is the total flow time given by the method h, and F_* is the minimum total flow time given by all compared methods, for a given test problem.

5 Analysis of Results

The ECS-FS performance was evaluated comparing its results with the two ant colony based algorithms (M-MMAS and PACO) shown by Rajendran and Ziegler [25], and the particle swarm method (PSO) shown by Tasgetiren et al. [30].

Nine test problems classes were considered, each one having ten instances. The classes were defined according to the number of tasks n being equal to 20, 50 and 100, and for each one of these values, the number of machines m being equal to 5, 10 and 20.

Table 1 presents the best solution obtained by the methods for each instance and shows the superiority of ECS-FS to the others for the test problems. The listed results of ECS-FS are the best out of 10 repeats. For a total of 90 instances the ECS-FS produced equal or better solutions for 82 of them, corresponding to 91.1%, and 59 solutions (65.5%) were inedited, better than the previous best found in the literaure.

Table 2 presents success rates for problems classes and shows that the ECS-FS had its success rate varying from 70% to 100%. This table shows also that the difference from the ECS-FS average relative deviation to the other method deviation is emphasized, reinforcing the proposed method superiority.

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.

Average processing time had a large variation from 8.62 seconds for the smallest problems class, with 20 tasks and 5 machines, to 7 hours and 39 minutes for the largest problems class used in this work, with 100 tasks and 20 machines.

6 Conclusion

The main objective of this work was apply CS to the Permutation Flow Shop Scheduling Problem of original and inedited form. Experimental results presented in the tables have shown that the ECS-FS method had superior performance regarding success rate and average relative deviation, compared with the best results found in the literature for the considered Flow Shop 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 Flow Shop has been the object of intense research in the last 50 years. 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 with better efficiency and solution quality, taking into account that the problem is NP-hard.

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.

Table 1. New Best Known solution for Taillard's benchmarks for Flowtime minimisation in Permutation Flow Shop

n	m	$M-MMAS$	$PACO$	PSO_{vns}	$ECS-FS$
20	5	14056	14056	**14033**	**14033**
		15151	15214	**15151**	**15151**
		13416	13403	**13301**	**13301**
		15486	15505	**15447**	**15447**
		13529	**13529**	**13529**	**13529**
		13139	**13123**	**13123**	**13123**
		13559	13674	**13548**	**13548**
		13968	14042	**13948**	**13948**
		14317	14383	**14295**	**14295**
		12968	13021	**12943**	**12943**
20	10	20980	20958	**20911**	**20911**
		22440	22591	**22440**	**22440**
		19833	19968	**19833**	**19833**
		18724	18769	**18710**	**18710**
		18644	18749	**18641**	**18641**
		19245	**19245**	19249	**19245**
		18376	18377	**18363**	**18363**
		20241	20377	**20241**	**20241**
		20330	**20330**	**20330**	**20330**
		21320	21323	**21320**	**21320**
20	20	**33623**	**33623**	34975	**33623**
		31604	31597	32659	**31587**
		33920	34130	34594	**33920**
		31698	31753	32716	**31661**
		34593	34642	35455	**34557**
		32637	32594	33530	**32564**
		33038	**32922**	33733	**32922**
		32444	32533	33008	**32412**
		33625	33623	34446	**33600**
		32317	32317	33281	**32262**
50	5	65768	65546	65058	**64838**
		68828	68485	68298	**68159**
		64166	64149	63577	**63453**
		69113	69359	68571	**68310**
		70331	70154	69698	**69477**
		67563	67664	67138	**66902**
		67014	66600	**66338**	66355
		64863	65123	64638	**64471**
		63735	63483	63227	**63068**
		70256	69831	69195	**69092**
50	10	89599	88942	88031	**87683**
		83612	84549	83624	**83535**
		81655	81338	80609	**80365**
		87924	88014	87053	**86934**
		88826	87801	87263	**86865**
		88394	88269	87255	**86969**
		90686	89984	**89259**	89304
		88595	88281	**87192**	87316
		86975	86995	**86102**	86213
		89470	89238	88631	**88534**
50	20	127348	126962	128622	**126315**
		121208	121098	122173	**119502**
		118051	117524	118719	**116910**
		123061	122807	123028	**121031**
		119920	119221	121202	**118914**
		122369	122262	123217	**121087**
		125609	125351	125586	**123340**
		124543	124374	125714	**123005**
		124059	123646	124932	**122203**
		126582	125767	126311	**124785**
50	20	127348	126962	128622	**126315**
		121208	121098	122173	**119502**
		118051	117524	118719	**116910**
		123061	122807	123028	**121031**
		119920	119221	121202	**118914**
		122369	122262	123217	**121087**
		125609	125351	125586	**123340**
		124543	124374	125714	**123005**
		124059	123646	124932	**122203**
		126582	125767	126311	**124785**
100	5	257025	257886	**254762**	254911
		246612	246326	245315	**243943**
		240537	241271	239777	**239002**
		230480	230376	**228872**	228888
		243013	243457	242245	**241659**
		236225	236409	**234082**	234172
		243935	243854	242122	**241753**
		234813	234579	232755	**232315**
		252384	253325	249959	**249608**
		246261	246750	244275	**244210**
100	10	305004	305376	303142	**301176**
		279094	278921	277109	**276902**
		297177	294239	292465	**290844**
		306994	306739	304676	**304377**
		290493	289676	288242	**287545**
		276449	275932	272790	**272635**
		286545	284846	282440	**282381**
		297454	297400	**293572**	294119
		309664	307043	305605	**304964**
		296869	297182	295173	**294362**
100	20	373756	372630	374351	**371391**
		383614	381124	379792	**376383**
		380112	379135	378174	**374599**
		380201	380765	380899	**378550**
		377268	379064	376187	**374426**
		381510	380464	379248	**377567**
		381963	382015	380912	**378367**
		393617	393075	392315	**389680**
		385478	380359	382212	**380152**
		387948	388060	386013	**383928**

Table 2. Success Rate (a) and Average Relative Deviation (b)

n	m	$M-MMAS$	$PACO$	PSO_{vns}	$ECS-FS$
20	5	20[a]	20	**100**	**100**
		0.1975[b]	0.4544	**0.0000**	**0.0000**
20	10	60	20	90	**100**
		0.0492	0.3235	0.0021	**0.0000**
20	20	20	20	0	**100**
		0.1195	0.1892	2.8278	**0.0000**
50	5	0	0	10	**90**
		1.1302	0.9450	0.2452	**0.0026**
50	10	0	0	30	**70**
		1.4196	1.1569	0.1841	**0.0322**
50	20	0	0	0	**100**
		1.2852	0.9780	1.8421	**0.0000**
100	5	0	0	30	**70**
		0.8733	0.9921	0.1638	**0.0104**
100	10	0	0	10	**90**
		1.2714	0.9834	0.2189	**0.0186**
100	20	0	0	0	**100**
		1.0678	0.8361	0.6627	**0.0000**

References

1. Ahmadi, R. H. and Bagchi, U.: Improved lower bounds for minimizing the sum of completion times of n jobs over m machines. European Journal of Operational Research. **44** (1990) 331–336
2. Allahverdi, A. and Aldowaisan, T.: New heuristics to minimize total completion time in m-machine flowshops. International Journal of Production Economics. **77** 71–83
3. Bean, J. C.: Genetic algorithm and random keys for sequencing and optimization. ORSA Journal on Computing. **6** (1994) 154–160
4. Campbell, H. G., Dudek, R. A. and Smith, M. L.: A heuristic algorithm for n-job, m-machine sequencing problem. Management Science. **16** (1970) 630–637
5. Cotta, C., Troya, J. M.: Genetic Forma Recombination in Permutation Flowshop Problems. Evolutionary Computation. **6** (1998) 25–44
6. Cotta, C., Fernandez, A. J.: Memetic Algorithms in Planning, Scheduling and Timetabling. Evolutionary Schedulling. Springer-Verlag (2006) 1–30
7. Framinan J. M., Leisten, R. and Ruiz-Usano R.: Comparison of heuristics for flowtime minimisation in permutation flowshops. Computer and Operations Research. **32** (2005) 1237–1254
8. Framinan, J. M. and Leisten, R.: An efficient constructive heuristic for flowtime minimisation in permutation flow shop. OMEGA. **31** (2003) 311–317
9. Garey, M. R., Johnson, D. S. and Sethi, R.: The Complexity of flowshop and jobshop scheduling. Mathematics of Operations Research. **1** (1976) 117–129
10. Glover, F.: Tabu search and adaptive memory programing: Advances, applications and challenges. In: Interfaces in Computer Science and Operations Research. Kluwer. (1996) 1–75
11. Gupta, J. N. D.: Heuristic algorithms for multistage flowshop scheduling problem. AIIE Transactions. **4** (1972) 11–18
12. Ho, J. C.: Flowshop sequencing with mean flowtime objective. European Journal of Operational Research. **81** (1995) 571–578

13. Kleeman, M. P. and Lamont, G. B.: Scheduling of flow-shop, job-shop, and combined scheduling problems using MOEAs with fixed and variable length chromosomes. Evolutionary Schedulling. Springer-Verlag (2006) 49–100
14. Li, X., Wang, Q. and Wu, C.: Efficient composite heuristics for total flowtime minimization in permutation flow shops. Omega. (2006) doi: 10.1016-j.omega.2006.11.003
15. Liu, J. and Reeves C. R.: Constructive and composite heuristic solutions to the $P//\sum C_i$ scheduling problem. European Journal of Operational Research. **132** (2001) 439-452
16. Merkle, D. and Middendorf, M.: An ant algorithm with a new pheromone evaluation rule for total tardiness problems. In: Lecture Notes in Computer Science, Vol. 1803. Springer-Verlag, Berlin (2000) 287–296
17. Miyazaki, S., Nishiyama, N. and Hashimoto, F.: An adjacent pairwise approach to the mean flowtime scheduling problem. Journal of the Operations Research Society of Japan. **21** (1978) 287–299
18. Nagano, M. S. and Moccellin, J. V.: Reducing mean flow time in permutation flow shop. Journal of the Operational Research Society (2007) doi: 10.1057/palgrave.jors.2602395
19. 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
20. 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
21. Oliveira, A. C. M. and Lorena, L. A. N.: Hybrid evolutionary algorithms and clustering search. In: Springer SCI Series, Crina Grosan, Ajith Abraham and Hisao Ishibuchi (eds) (2007) acepted (http://www.lac.inpe.br/~lorena/alexandre/HEA-07.pdf)
22. Rajendran, C. and Chaudhuri, D.: An efficient heuristic approach to the scheduling of jobs in a flowshop. European Journal of Operational Research. **61** (1991) 318–325
23. Rajendran, C.: Heuristic algorithm for scheduling in a flowshop to minimise total flowtime. International Journal Production Economics. **29** (1993) 65–73
24. Rajendran, C. and Ziegler, H.: An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. European Journal of Operational Research. **103** (1997) 129–138
25. Rajendran, C. and Ziegler H.: Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research. **155** (2004) 426–438
26. Rinnooy Kan, A. H. G.: Machine Scheduling Problems: Classification, Complexity, and Computations. The Hahue: Nijhoff (1976)
27. Syswerda, G.: Uniform crossover in genetic algorithms. In: International Conference on Genetic Algorithms (ICGA). Virginia, USA, (1989) 2–9
28. Stutzle, T.: Applying iterated local search to the permutation flowshop problem. Technical Report, AIDA-98-04, Darmstad University of Technology, Computer Science Department, Intelletics Group, Darmstad, Germany. (1998)
29. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research. **64** (1993) 278–285
30. Tasgetiren, M. F., Liang, Y. C., Sevkli, M. and Gencyilmaz, G.: A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. European Journal of Operational Research. **177** (2007) 1930–1947

31. Wang, C., Chu, C. and Proth, J. M.: Heuristic approaches for n/m/F/ Ci scheduling problems. European Journal of Operational Research. **96** (1997) 636–644
32. Watson, J. P., Barbulescu, L., Whitley L. D. and Howe, A. E.: Contrasting structured and random permutation flowshop scheduling problems: Search space topology and algorithm performance. ORSA Journal of Computing. **14** (2002) 98–123
33. Woo, D. S. and Yim H. S.: A heuristic algorithm for mean flowtime objective in flowshop scheduling. Computers and Operations Research. **25** (1998) 175–182