Applied Soft Computing 10 (2010) 1293–1300

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Multiprocessor task scheduling in multistage hybrid flow-shops: A parallel greedy algorithm approach

Cengiz Kahraman[a], Orhan Engin[b], İhsan Kaya[a,b,*], R. Elif Öztürk[b]

[a] Department of Industrial Engineering, Istanbul Technical University, 34367 Istanbul, Turkey
[b] Department of Industrial Engineering, Faculty of Engineering and Architecture, University of Selçuk, Konya 42079, Turkey

ARTICLE INFO

Article history:
Received 31 March 2008
Received in revised form
14 November 2009
Accepted 19 March 2010
Available online 27 March 2010

Keywords:
Hybrid flow shop
Multiprocessor tasks scheduling problems
Parallel greedy algorithm

ABSTRACT

Hybrid flow shop scheduling problems have a special structure combining some elements of both the flow shop and the parallel machine scheduling problems. Multiprocessor task scheduling problem can be stated as finding a schedule for a general task graph to execute on a multiprocessor system so that the schedule length can be minimized. Hybrid Flow Shop Scheduling with Multiprocessor Task (HFSMT) problem is known to be NP-hard. In this study we present an effective parallel greedy algorithm to solve HFSMT problem. Parallel greedy algorithm (PGA) is applied by two phases iteratively, called destruction and construction. Four constructive heuristic methods are proposed to solve HFSMT problems. A preliminary test is performed to set the best values of control parameters, namely population size, subgroups number, and iteration number. The best values of control parameters and operators are determined by a full factorial experimental design using our PGA program. Computational results are compared with the earlier works of Oğuz et al. [1,3], and Oğuz [2]. The results indicate that the proposed parallel greedy algorithm approach is very effective in terms of reduced total completion time or makespan ($C_{\max}$) for the attempted problems.

1. Introduction

Hybrid flow shop (HFS) scheduling problems combine the properties of flow-shop and parallel machine scheduling problems. In the classical flow shop, there is only one processor (machine) at each stage, and jobs visit the stages of the shop in the same order of machines. But in the HFS, there are one or more uniform, identical and unrelated parallel processors at each stage. In HFS scheduling problems, it is assumed that a set $J=\{1, 2, \ldots, n\}$ of n jobs exist and each is to be processed on a set of k-stage flow shop. In each stage i ($i=1, 2, \ldots, k$), there are m identical machines in parallel. The schematic presentation of a HFS scheduling problem is given in Fig. 1 where M_{mk} represents the machine in the mth parallel and kth stage.

The two-stage HFS scheduling problem is NP-hard, even if there is only one machine on the first stage and two machines on the second stage [4].

Recently, the HFS scheduling problem has received much attention due to its theoretical and practical importance. HFS scheduling has been widely applied in different manufacturing environments like iron-steel, manufacturing, textile, machine driven and electronics industries. Most of the research on HFS scheduling concentrates on two-stage problems [5]. The first model on HFS has been proposed by Arthanari and Ramamurthy [6]. They developed a Branch & Bound (B&B) method for HFS. Brah and Hunsucker [7] considered a flow shop with multiple processors for at least one stage. They also proposed a B&B algorithm to minimize the makespan.

Hoogeveen et al. [8] showed that preemptive scheduling in a two-stage flow shop with at least two identical parallel machines is NP-hard in the strong sense. Portmann et al. [5] improved the lower bound values of that problem. They provided a hybrid algorithm crossing B&B with genetic algorithm (GA). Riane et al. [9] treated the problem of scheduling n jobs on a three-stage hybrid flow shop of particular structure and proposed two heuristic procedures to cope with the realistic problems. Grangeon et al. [10] proposed a generic simulation model for HFS where the job priorities at each machine stage are established dynamically. Moursli and Pochet [11] developed a new B&B algorithm which reduced the initial gap between upper and lower bounds to half it in a few minutes of running time. Negenman [12] provided a local search method to solve HFS problems. Linn and Zhang [13] reviewed the state-of-the-art in HFS scheduling. Recently, the metaheuristic methods have been proposed for this problem. Engin and Döyen [14] presented artificial immune systems for HFS scheduling problems. Tang et al. [15] proposed a neural network model and algorithm to solve the dynamic HFS problem. Zandieh et al. [16] proposed an immune algorithm approach for scheduling a sequence-dependent setup times hybrid flow shop. Allaoui and Artiba [17] handled a two

* Corresponding author. Tel.: +90 212 293 13 00x2766; fax: +90 212 240 72 60.
E-mail address: kayai@itu.edu.tr (İ. Kaya).

1568-4946/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2010.03.008

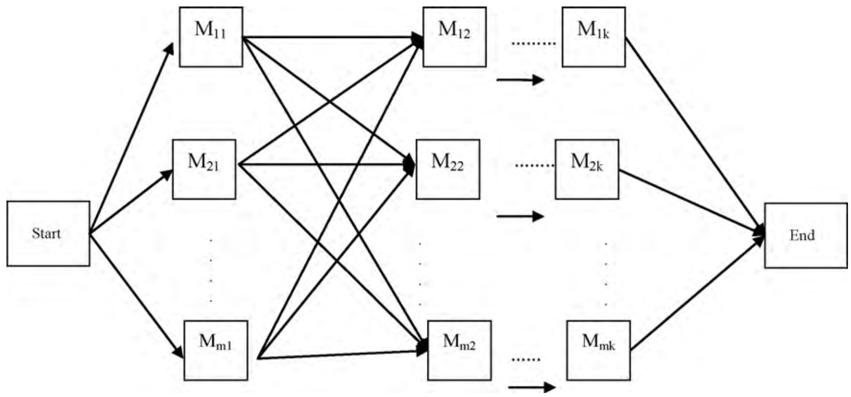

Fig. 1. A hybrid flow shop.

stages HFS scheduling problem with only one machine on the first stage and m machines on the second stage to minimize the makespan. They used a B&B model for this problem. Haouari et al. [18] presented an exact Branch and Bound algorithm for the two stage hybrid flow shop problem with multiple identical machines in each stage. The objective is to schedule a set of jobs so as to minimize the makespan. Caricato et al. [19] used a variant of the TSP problem for solving a batch-wise HFS. The effectiveness of the proposed approach is validated through a comparison with different heuristics traditionally used in HFS scheduling problems. Janiak et al. [20] studied the flow shop with parallel machines at each stage (machine center). The scheduling criterion consists of three parts; the total weighted earliness, the total weighted tardiness and the total weighted waiting time. To solve the problem, they developed three constructive algorithms and three metaheuristic methods. Vob and Witt [21] considered a real-world multi-mode multi-project scheduling problem in which the resources formed a hybrid flow shop consisting of 16 production stages. Alaykıran et al. [22] presented an ant colony optimization model to solve HFS problems.

The multiprocessor task scheduling is a relatively new area of HFS problems [23]. In HFS problems, jobs require only a single processor to be processed. This restriction can be relaxed to allow for multiprocessor tasks [24].

In this paper, the multiprocessor task scheduling in a hybrid flow shop is considered. In the HFSMT problem, each task, within a job, requires one or several processors (machines) simultaneously at each stage [1]. A HFSMT problem can be stated formally as follows: consider a set of n jobs to be processed in a k-stage flow-shop, where each stage i has m_i identical parallel processors (machines) [3]. It is convenient to view a job as a sequence of k tasks, one task for each stage, where the processing of any task can commence only after the completion of the preceding task [1]. Each task within a job requires one or several processors simultaneously [1]. Let $size_{ij}$ be the number of the processors required to process the job j at stage i and P_{ij} be the processing time of the job j at stage i. In other words, the task i of job j has to be processed on $size_{ij}$ of m_i that are identical parallel processors of stage i for p_{ij} time units [3]. This problem can be denoted by $F_m(P_{m1}, \ldots, P_{mm})/size_{ij}/C_{max}$. The terms $F_m(P_{m1}, \ldots, P_{mm})$, $size_{ij}$ and C_{max} describe the machine environment, details of processing characteristics, and the objective to be minimized, respectively. If we have $size_{ij} = 1$ for all, $F_m(P_{m1}, \ldots, P_{mm})/size_{ij}/C_{max}$ (HFSMT) problem will be reduced to the classical HFS makespan minimization problem.

For these types of problems, the following assumptions are made [3]:

- The number of jobs and machines at each stage and processing times are known and fixed.
- The number of stages and processors at each stage are known and fixed.
- The setup times of the tasks are included in the processing times.
- All the jobs and processors are available at the beginning of the scheduling period.
- Each processor can process at most one task at the same time.

In the literature, several search algorithms are proposed for this problem. Edwin et al. [25] proposed a GA for the multiprocessor scheduling problem. Oğuz and Ercan [26]; Oğuz et al. [27] developed constructive heuristic algorithms to schedule the unit processing time multiprocessor tasks in a two-stage HFS for minimizing the makespan. Oğuz et al. [1] proposed a Tabu Search (TS) algorithm to solve HFSMT problems. Şerifoğlu and Tiryaki [28] developed a simulated annealing algorithm to solve HFSMT problems. Şerifoğlu and Ulusoy [24] proposed a GA for multiprocessor task scheduling in a multi-stage hybrid flow shop scheduling problem. Oğuz and Ercan [3] also developed a GA for the hybrid flow shop scheduling with multiprocessor task problems. Kwok and Ahmad [29] proposed optimal algorithms for static scheduling of task graphs with arbitrary parameters to multiple homogeneous processors. Zinder et al. [30] concerned with scheduling problems with multiprocessor tasks and they presented the conditions under which such problems can be solved in polynomial time. Coll et al. [31] considered the problem of scheduling a set of tasks related to precedence constraints to a set of processors, so as to minimize their makespan. Shenassa and Mahmoodi [32] proposed a novel intelligent solution based on a GA and chromosome background tree for task scheduling in multiprocessor systems. Ying and Lin [23] proposed a novel ant colony system to solve multiprocessor task scheduling in multistage hybrid flow-shop. Kuszner and Malafiejski [33] considered a problem of preemptive scheduling of multiprocessor tasks on dedicated to processors in order to minimize the sum of completion times. Cheng et al. [34] presented a GA based approach combined with a feasible energy function for multiprocessor scheduling problems with resource and timing constraints in dynamic real time scheduling. Recently, Hwang et al. [35] described the multiprocessor scheduling problem based on a deterministic model. The execution time of tasks and the communication costs between tasks were taken into account. They proposed the extension of the priority-based coding method by using the priority based multi-chromosome in genetic algorithms (GAs) to solve the problem. Tseng and Liao [36] analyzed the multistage HFS scheduling problems with multiprocessor tasks by using particle swarm optimization (PSO). They also compared the results of PSO algorithm with two existing methods: GA and ant colony system (ACS) algorithm. The results showed that the proposed PSO algorithm outperformed all the existing algorithms for the considered problem. Ying [37] proposed a simple iterated greedy (IG) heuristic to minimize makespan in a multistage HFS with multiprocessor tasks. Ying [37] managed computational experiments on two well-known benchmark problem sets to validate and verify the proposed heuristic. The experiment results showed that the proposed IG heuristic was highly effective as compared to three state-of-the-art meta-heuristics on the same benchmark instances.

In this paper, an effective PGA is developed to solve HFSMT problems. The effectiveness of the developed method is tested with the earlier studies of Oğuz et al. [1,3], and Oğuz [2]. The computational results indicate that the developed approach is more effective for the attempted problems. To the best of our knowledge, there is no PGA applied to HFSMT problems in the literature.

The rest of the paper is organized as follows. The proposed PGA and the parameter optimization are described in Section 2. In Section 3, the computational results are presented. Conclusions of the paper and future research are emphasized in Section 4.

2. Parallel greedy algorithms

In the literature, a PGA was applied as a Parallel Greedy Randomized Adaptive Search Procedure (GRASP) to the scheduling problem by Binato et al. [38]. Aiex et al. [39] presented a new parallel GRASP with path-relinking for the Job Shop Scheduling (JSS) problems. Ruiz and Stützle [40] proposed an effective iterated greedy algorithm for permutation flow-shop scheduling problems. The greedy algorithm is applied to two phases iteratively, named destruction and construction for the flow-shop scheduling problem. Baraz and Mosheiov [41] proposed a greedy heuristic algorithm for flow shop makespan minimization with no machine idle-time problems.

Greedy algorithm is an iterative search process and it is relatively fast and an alternative approach for combinatorial optimization problems. It builds up a solution in small steps. There are two characteristics of a greedy algorithm [42]:

1. At each stage the best value is chosen without worrying whether it will be the best decision in the long run.
2. Once a decision is made, it is never reversed.

Running time of the greedy algorithm is $O(m \operatorname{Log} n)$ where n is the number of nodes and m is the number of edge [43].

There are a number of advantages to greedy algorithms. First, the amount of computation is known in advance and the computation is usually linear or quadratic with respect to the size of the problem. If a good criterion is selected for making the choice at each step, it can often generate good solutions [44]. Greedy algorithms are generally fast, easy to implement, and often provide very good solutions.

A general greedy algorithm for machine scheduling is given as follows [45]:

- *Input:*
  - *Jobs 1, . . ., n with job durations* $d_1, \ldots, d_n$.
  - *A number m of machines (typically,* $m \ll n$*).*
- *Output:A machine* $m_i \in \{1, \ldots, m\}$ *and a start time* $t_i \geq 0$ *for each job* $i \in \{1, \ldots, n\}$.
- *Constraint:Disjoint execution times for* $i, j \in \{1, \ldots, n\}$ *if* $m_i = m_j$*, that is, either* $t_i + d_i \leq t_j$ *or* $t_i + d_i \leq t_i$
- *Minimize:Total completion time,* $\max\{t_i + d_i \mid i = 1, \ldots, n\}$
- *Greedy procedure:*

  - *For* $j = 1, \ldots, m$*:* $max_j = 0$
  - *For* $i = 1, \ldots, n$ *(in this order) do:*

- *Let* $j \in \{1, \ldots m\}$ *such that* max_j *is minimal.*
- *Set* $m_i = j$ *and* $t_i = max_j$
- *Increase* max_j *by* d_i

Greedy algorithm is usually applied by two phases iteratively, called destruction and construction phases for scheduling problems. During the destruction phase, some jobs are eliminated from the incumbent solution and in the construction phase, the eliminated jobs are reinserted into the sequence using some constructive heuristic methods.

2.1. Destruction and construction phases

The destruction phase is applied to a permutation π of n jobs and it is chosen randomly without repetition of d jobs where d is the number of subgroups which consists of selected jobs. These d jobs are then removed from π in the order which they were chosen [40]. As the results of this phase, two subsequences are obtained. The first is the partial sequence π_D with n-d jobs, which is the sequence of d jobs. The second is a sequence of d jobs, which is denoted as π_R. This π_R contains the jobs that have to be reinserted into π_D to yield a complete candidate solution in the order which they were removed from π [40].

The construction phase starts with the subsequence π_D and performs d steps, which the jobs in π_R are reinserted into π_D. In this study four constructive heuristic methods are proposed. They are explained in the following:

2.1.1. One point step constructive methods

In one point step (OPS) constructive methods, the first job of π_R is inserted into the first position of π_D and then the second job of π_R is inserted into the third position of π_D. This process is iterated until π_R becomes empty.

2.1.2. Two point step constructive methods

In two point step (TPS) constructive methods, the first job of π_R is inserted into the first position of π_D and then the second job of π_R is inserted into the fourth position of π_D. This process is iterated until π_R becomes empty.

2.1.3. Initial position (IP) constructive methods

In the initial position of constructive methods, the π_R-sublists are initially inserted into the π_D.

2.1.4. End position (EP) constructive methods

In the end position constructive methods, the π_R-sublists are inserted into the end of the π_D.

In the following, we present an example to show the steps of these four constructive methods.

2.1.5. Example

Consider six jobs to be scheduled in a HFS scheduling. We assume that the initial job sequence is (1, 2, 3, 4, 5, 6) and it is obtained randomly. The π_R and π_D are randomly obtained as (2, 4, 5) and (1, 3, 6), respectively. The four constructive methods are illustrated in Fig. 2.

2.2. PGA for HFSMT scheduling problems

In this section an iterative parallel greedy algorithm for solving HFSMT problems with makespan ($C_{\max}$) criteria is proposed. The flowchart of the PGA approach is depicted in Fig. 3.

In the study, each solution is represented by a permutation of a sequence of jobs and this permutation of jobs is transformed to a solution for HFSMT by a Gantt chart.

The detailed steps are described as follows:

Step 0: Set the parameters:

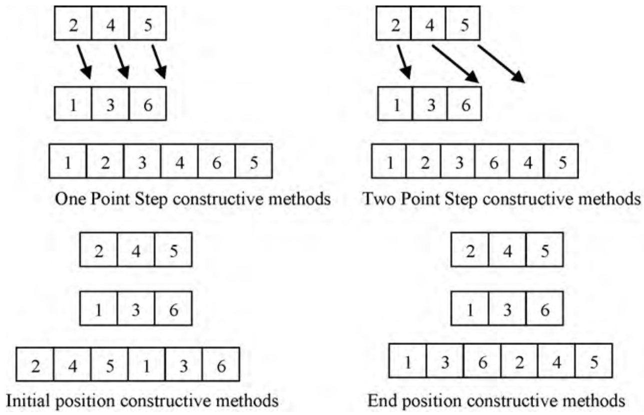

Fig. 2. Illustration of constructive methods.

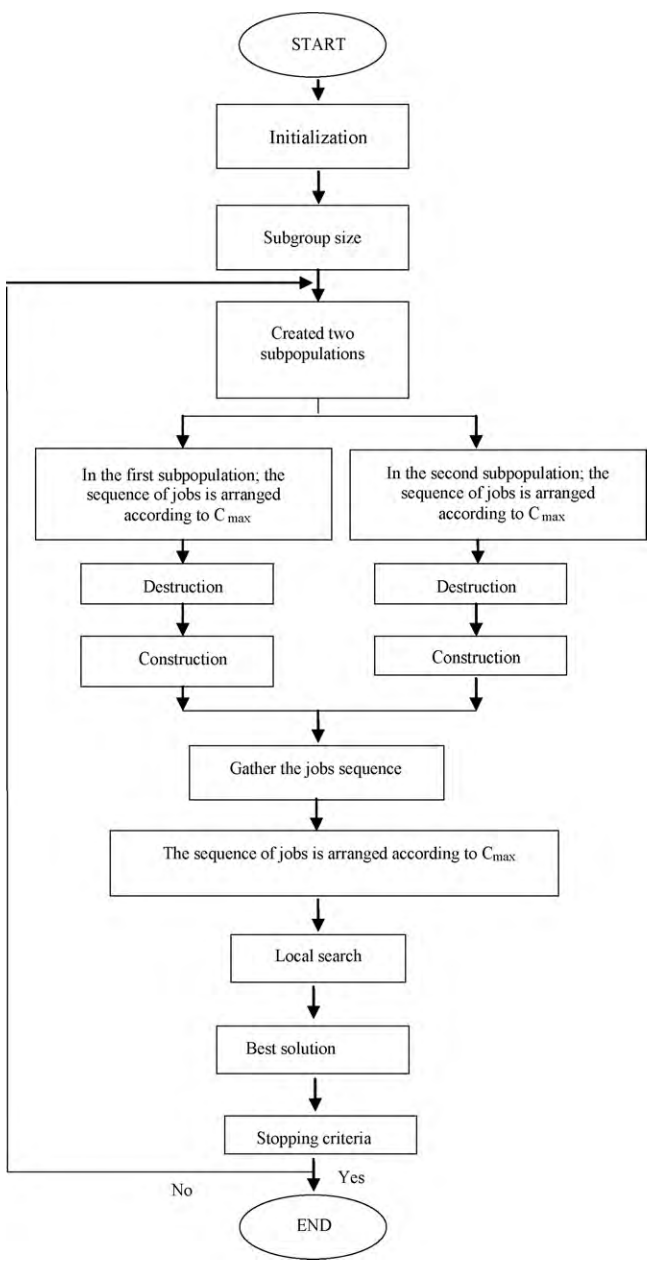

Fig. 3. The flowchart of the PGA approach.

Population size; number of subgroups (d); number of iterations.

Step 1: Initialize:

Generate an initial population randomly; determine the makespan of every job sequence.

Step 2: Subpopulations:

Create two subpopulations. A subpopulation is randomly generated solutions.

Step 3: Destruction:

Select π positions from one job sequence at random without repetition and then remove π positions (jobs) from the sequence.

Step 4: Construction:

Execute the best constructive heuristic method.

Step 5: Local search:

To improve each solution, use insertion neighborhood procedure.

Step 6: Best solution:

After the insertion neighborhood procedure, select the best solution of the makespan.

Step 7: Stopping criteria:

Choose the iteration number as 250 or 25 min CPU time.

2.2.1. Subgroup number

In the destruction procedure; d jobs are randomly chosen in the n jobs sequence, where d is showing the number of subgroups. Subgroup number has a very important role in PGA and it helps to maintain diversity in the population. Subgroup number can be generated from the range $(1;\ n-1)$. In this study subgroup number is tested for HFSMT problems using the range $(1;n-1)$. The initial population size is divided by two and thus two separate subpopulations with equal size are obtained.

For parallel calculating, exactly two subpopulations are chosen. The construction and destruction methods are applied to all subpopulations job sequence. In our study, the population size is accepted as the permutation (π) of n jobs. Local search algorithm is applied based on the insertion neighborhood, which is commonly regarded as a very good choice for the scheduling problem [38,44].

Table 1
The best parameters set of each benchmark problem.

Problem	Pop. size	Subgroup	Construction method
P10S2T01	30	8	OPS
P20S2T01	30	7	TPS
P50S2T01	30	2	TPS
P1HS2T01	15	2	EP
P10S5T01	30	3	TPS
P20S5T01	30	2	OPS
P50S5T01	30	3	EP
P1HS5T01	15	2	EP
P10S8T01	15	4	TPS
P20S8T01	15	2	EP
P50S8T01	15	4	EP
P1HS8T01	15	3	OPS
Q10S2T1	15	3	IP
Q20S2T1	30	8	EP
Q50S2T1	15	2	TPS
Q1HS2T1	15	4	TPS
Q10S5T1	15	2	OPS
Q20S5T1	30	4	OPS
Q50S5T1	15	4	OPS
Q1HS5T1	15	4	OPS
Q10S8T1	15	3	OPS
Q20S8T1	30	6	EP
Q50S8T1	30	2	EP
Q1HS8T1	15	4	EP

Table 2
The computational results.

Problem	GA	PGA		Problem	GA	PGA	
	APD	APD	CPU		APD	APD	CPU
P10S2T01	0.000	0.000	00 : 00.062	P10S5T01	4.530	4.878	00:04.844
P10S2T02	0.000	0.000	00 : 00.016	P10S5T02	6.022	6.022	00:00.000
P10S2T03	5.960	5.960	00 : 01.406	P10S5T03	7.540	7.540	00:01.984
P10S2T04	0.000	0.000	00 : 00.000	P10S5T04	8.609	11.755	00:01.469
P10S2T05	0.000	0.000	00 : 00.000	P10S5T05	3.684	3.684	00:00.360
P10S2T06	0.000	0.000	00 : 00.000	P10S5T06	0.000	0.000	00:00.000
P10S2T07	0.000	0.000	00 : 00.000	P10S5T07	8.210	8.210	00:02.063
P10S2T08	0.000	0.000	00 : 00.000	**P10S5T08**	**2.742**	**2.258**	**00:03.516**
P10S2T09	0.000	0.000	00 : 00.000	P10S5T09	9.076	9.076	00:00.040
P10S2T10	0.000	0.000	00 : 00.000	P10S5T10	10.417	12.179	00:00.000
P20S2T01	**1.997**	**1.690**	**00 : 00.015**	P20S5T01	1.567	1.567	00:01.610
P20S2T02	**2.368**	**1.275**	**00 : 02.516**	**P20S5T02**	**7.165**	**6.858**	**00:04.188**
P20S2T03	0.000	0.000	00 : 00.000	P20S5T03	2.561	6.170	00:00.062
P20S2T04	0.000	0.000	00 : 00.000	**P20S5T04**	**2.268**	**0.113**	**00:18.344**
P20S2T05	0.000	0.000	00 : 00.000	**P20S5T05**	**2.313**	3.049	00:00.031
P20S2T06	0.000	0.000	00 : 00.063	**P20S5T06**	**2.650**	**2.479**	**00:00.047**
P20S2T07	0.341	0.341	00 : 00.765	P20S5T07	0.000	0.000	00:01.891
P20S2T08	0.278	0.278	00 : 00.968	P20S5T08	3.519	3.519	00:00.000
P20S2T09	0.000	0.000	00 : 00.000	P20S5T09	0.000	0.000	00:00.000
P20S2T10	0.000	0.000	00 : 00.484	P20S5T10	7.165	7.165	00:44.784
P50S2T01	**1.753**	**1.088**	**00 : 13.265**	**P50S5T01**	**0.911**	**0.835**	**02:05.547**
P50S2T02	0.000	0.000	00 : 00.000	**P50S5T02**	**4.539**	**0.532**	**00:33.921**
P50S2T03	**0.590**	**0.215**	**01 : 06.391**	P50S5T03	0.998	0.998	00:44.484
P50S2T04	1.238	1.423	01 : 39.297	P50S5T04	0.618	4.325	00:00.704
P50S2T05	0.353	0.588	00 : 00.078	**P50S5T05**	**2.577**	**0.000**	**00:04.781**
P50S2T06	0.000	0.000	00 : 00.016	P50S5T06	0.000	0.000	00:01.734
P50S2T07	**2.716**	**2.130**	**00 : 47.172**	**P50S5T07**	**1.100**	**0.477**	**01:47.234**
P50S2T08	0.000	0.000	00 : 00.015	**P50S5T08**	**2.447**	**0.745**	**00:34.812**
P50S2T09	0.000	0.000	00 : 00.047	**P50S5T09**	**5.096**	**0.668**	**01:21.672**
P50S2T10	0.262	0.052	00 : 03.891	P50S5T10	0.271	1.264	01:07.375
P1HS2T01	**0.534**	**0.507**	**02 : 55.437**	**P1HS5T01**	**3.493**	**0.056**	**00:50.516**
P1HS2T02	0.000	0.000	00 : 00.016	**P1HS5T02**	**1.543**	0.000	00:13.265
P1HS2T03	**0.814**	**0.603**	**01 : 08.890**	**P1HS5T03**	**3.819**	**2.272**	**01:10.688**
P1HS2T04	0.000	0.000	00 : 00.172	P1HS5T04	1.425	1.548	00:00.156
P1HS2T05	0.000	0.000	00 : 00.156	**P1HS5T05**	**2.347**	**0.000**	**00:11.531**
P1HS2T06	0.000	0.000	00 : 00.031	P1HS5T06	3.591	4.488	00:00.266
P1HS2T07	**0.926**	**0.538**	**02 : 03.328**	**P1HS5T07**	**0.300**	**0.000**	**00:16.094**
P1HS2T08	**0.020**	**0.000**	**00 : 00.172**	**P1HS5T08**	**0.673**	**0.000**	**00:17.469**
P1HS2T09	0.105	0.394	00 : 00.188	**P1HS5T09**	**1.379**	**0.000**	**01:31.640**
P1HS2T10	0.097	0.242	01 : 47.734	P1HS5T10	3.248	3.389	00:00.375
P10S8T01	21.268	23.662	00:58.907	Q10S2T1	0.000	0.000	00:00.000
P10S8T02	26.179	28.455	00:00.000	Q10S2T2	5.012	5.012	00:00.000
P10S8T03	20.027	21.786	00:31.345	**Q10S2T3**	**9.859**	**0.000**	**00:00.234**
P10S8T04	13.091	14.038	00:00.000	Q10S2T4	0.000	0.000	00:00.000
P10S8T05	1.902	4.212	00:01.392	Q10S2T5	0.000	19.209	00:00.000
P10S8T06	0.409	0.409	00:00.000	Q10S2T6	5.817	5.817	00:00.000
P10S8T07	24.757	26.214	00:00.000	Q10S2T7	0.286	0.286	00:00.343
P10S8T08	19.479	21.933	00:00.000	Q10S2T8	12.333	93.33	00:00.000
P10S8T09	0.830	2.075	00:00.000	Q10S2T9	4.076	4.076	00:00.078
P10S8T10	19.589	20.000	00:28.562	Q10S2T10	7.331	7.331	00:00.062
P20S8T01	8.163	9.448	00:00.125	Q20S2T1	7.193	7.193	00:00.125
P20S8T02	0.000	0.000	00:10.813	Q20S2T2	0.892	35.414	00:00.000
P20S8T03	2.573	4.117	01:21.219	Q20S2T3	4.690	46.529	00:00.000
P20S8T04	1.593	3.805	00:00.188	Q20S2T4	0.000	0.000	00:00.125
P20S8T05	**3.152**	**2.641**	**00:00.234**	**Q20S2T5**	**0.844**	**0.703**	**00:00.032**
P20S8T06	8.163	9.675	00:00.141	Q20S2T6	0.426	0.426	00:01.890
P20S8T07	23.795	27.487	00:00.235	Q20S2T7	0.000	0.000	00:00.000
P20S8T08	**3.791**	**2.729**	**00:37.093**	Q20S2T8	0.369	0.369	00:00.000
P20S8T09	0.000	0.000	00:04.828	Q20S2T9	0.000	0.000	00:00.000
P20S8T10	4.117	4.803	00:00.031	**Q20S2T10**	**1.826**	**1.674**	**00:00.047**
P50S8T01	2.709	6.578	00:00.172	Q50S2T1	0.161	0.161	00:00.078
P50S8T02	2.750	2.750	00:29.516	Q50S2T2	0.329	0.329	00:13.259
P50S8T03	0.936	1.161	00:00.359	Q50S2T3	1.651	4.068	00:00.062
P50S8T04	**4.760**	**3.917**	**00:00.171**	**Q50S2T4**	**0.623**	**0.187**	**00:15.484**
P50S8T05	**3.639**	**2.691**	**00:13.594**	Q50S2T5	0.000	0.000	00:01.046
P50S8T06	**1.875**	**1.641**	**00:50.234**	Q50S2T6	2.353	21.799	00:14.640
P50S8T07	**5.436**	**3.701**	**00:33.453**	Q50S2T7	0.000	0.000	00:00.109
P50S8T08	**2.434**	**1.917**	**00:11.828**	Q50S2T8	3.292	7.081	00:00.204
P50S8T09	**4.760**	**4.465**	**00:31.813**	Q50S2T9	2.747	2.816	00:23.153
P50S8T10	**5.371**	**4.726**	**00:16.734**	Q50S2T10	2.567	9.949	00:00.141

Table 2 (*Continued*)

Problem	GA	PGA		Problem	GA	PGA	
	APD	APD	CPU		APD	APD	CPU
P1HS8T01	**2.877**	**1.509**	**01:25.531**	**Q1HS2T1**	**2.250**	**2.179**	**00:24.422**
P1HS8T02	**3.568**	**0.801**	**00:31.610**	**Q1HS2T2**	**1.904**	**1.666**	**00:00.203**
P1HS8T03	**1.987**	**1.472**	**00:00.000**	Q1HS2T3	3.849	3.952	00:24.469
P1HS8T04	**2.149**	**0.498**	**00:05.735**	Q1HS2T4	0.000	5.008	00:00.500
P1HS8T05	**1.944**	**0.731**	**00:50.985**	Q1HS2T5	0.029	0.029	00:23.672
P1HS8T06	**3.422**	**2.907**	**00:47.750**	Q1HS2T6	0.911	1.058	00:00.265
P1HS8T07	**2.426**	**1.203**	**02:18.532**	Q1HS2T7	1.891	2.247	01:18.422
P1HS8T08	7.294	8.451	00:00.609	Q1HS2T8	0.118	0.355	00:28.438
P1HS8T09	**1.951**	**1.334**	**00:57.891**	**Q1HS2T9**	**1.642**	**0.000**	**02:58.468**
P1HS8T10	**3.838**	**0.985**	**00:36.500**	Q1HS2T10	1.885	1.885	01:19.234
Q10S5T1	9.291	9.291	00:00.563	Q10S8T1	15.860	25.134	00:03.781
Q10S5T2	1.536	1.877	00:01.140	**Q10S8T2**	**7.570**	**2.789**	**00:23.875**
Q10S5T3	5.664	7.434	00:00.000	**Q10S8T3**	**8.005**	**7.349**	**00:01.125**
Q10S5T4	6.919	14.992	00:01.503	**Q10S8T4**	**2.289**	**2.169**	**00:09.110**
Q10S5T5	2.574	2.574	00:16.125	Q10S8T5	19.635	20.700	00:01.031
Q10S5T6	8.155	10.485	00:00.000	Q10S8T6	13.018	17.456	00:02.250
Q10S5T7	15.146	15.146	00:00.500	Q10S8T7	14.944	32.432	00:01.718
Q10S5T8	**3.811**	**0.000**	**00:02.734**	Q10S8T8	13.580	14.938	00:20.209
Q10S5T9	0.000	0.000	00:00.875	Q10S8T9	18.053	19.937	00:31.725
Q10S5T10	13.163	13.163	00:00.766	**Q10S8T10**	**7.261**	**4.331**	**00:08.562**
Q20S5T1	**3.533**	**2.257**	**00:58.781**	**Q20S8T1**	**11.168**	**10.656**	**00:00.015**
Q20S5T2	7.970	7.970	00:00.063	Q20S8T2	18.653	21.634	00:00.234
Q20S5T3	**13.626**	**13.626**	**00:00.047**	Q20S8T3	10.169	10.829	00:00.109
Q20S5T4	4.277	9.364	00:00.078	**Q20S8T4**	**9.436**	**9.347**	**00:31.625**
Q20S5T5	**13.907**	**0.000**	**00:39.687**	Q20S8T5	18.469	19.898	00:00.250
Q20S5T6	7.093	10.930	00:00.047	**Q20S8T6**	**11.168**	**10.143**	**00:00.047**
Q20S5T7	4.580	4.580	00:09.828	**Q20S8T7**	**40.285**	**24.808**	**00:48.328**
Q20S5T8	**11.578**	**11.323**	**00:00.109**	Q20S8T8	21.613	22.366	00:18.977
Q20S5T9	1.647	3.074	00:00.031	Q20S8T9	20.128	18.522	00:00.063
Q20S5T10	**5.345**	**5.122**	**00:00.016**	**Q20S8T10**	**14.924**	**13.909**	**00:00.047**
Q50S5T1	**15.558**	**13.707**	**01:30.344**	**Q50S8T1**	**20.491**	**19.151**	**00:00.140**
Q50S5T2	**9.406**	**7.839**	**00:00.125**	Q50S8T2	14.607	15.236	00:00.609
Q50S5T3	**10.042**	**0.000**	**00:56.000**	**Q50S8T3**	**9.231**	**1.921**	**01:06.516**
Q50S5T4	**11.830**	**0.100**	**00:54.079**	**Q50S8T4**	**24.140**	**22.742**	**00:00.000**
Q50S5T5	**17.784**	**15.685**	**00:26.532**	**Q50S8T5**	**22.415**	**14.016**	**00:12.875**
Q50S5T6	**12.231**	**11.665**	**00:00.203**	**Q50S8T6**	**13.731**	**13.068**	**00:00.313**
Q50S5T7	**8.294**	**3.622**	**00:42.891**	**Q50S8T7**	**9.445**	**4.035**	**00:45.797**
Q50S5T8	**11.680**	**0.359**	**01:02.594**	Q50S8T8	24.389	31.505	00:00.390
Q50S5T9	**2.479**	**0.000**	**00:41.203**	Q50S8T9	14.828	16.624	00:00.218
Q50S5T10	**2.723**	**1.733**	**00:46.906**	**Q50S8T10**	**15.652**	**8.807**	**01:04.078**
Q1HS5T1	**18.948**	**10.089**	**00:33.641**	**Q1HS8T1**	**10.581**	**9.298**	**00:00.656**
Q1HS5T2	**15.451**	**0.000**	**00:57.797**	**Q1HS8T2**	**8.463**	**6.153**	**01:32.781**
Q1HS5T3	**16.207**	**2.502**	**03:41.358**	**Q1HS8T3**	**14.856**	**14.691**	**04:24.343**
Q1HS5T4	**18.948**	**11.750**	**02:01.359**	**Q1HS8T4**	**16.203**	**12.035**	**00:55.281**
Q1HS5T5	**24.866**	**13.658**	**01:21.000**	**Q1HS8T5**	**15.998**	**15.157**	**01:25.203**
Q1HS5T6	**17.349**	**7.610**	**00:21.656**	**Q1HS8T6**	**16.969**	**16.601**	**00:01.391**
Q1HS5T7	**24.796**	**9.139**	**01:07.140**	**Q1HS8T7**	**15.692**	**13.286**	**01:03.172**
Q1HS5T8	**18.322**	**4.895**	**01:20.500**	Q1HS8T8	15.458	17.952	02:56.815
Q1HS5T9	**21.104**	**3.276**	**01:33.938**	**Q1HS8T9**	**11.234**	**2.808**	**00:27.203**
Q1HS5T10	**22.710**	**8.952**	**01:03.172**	Q1HS8T10	9.151	9.710	01:33.675

2.3. Parameter optimization for PGA

The PGA's efficiency depends upon the selection of the control parameters. These parameters are population size, subgroups number, construction method, iteration number or stopping criteria. The determination of suitable settings for the control parameters of any metaheuristic method is a very difficult task. The PGA search process is controlled by multiple factors (control parameters) whose effects may possibly interact. In general there are a few control mechanisms for these parameters. We used full factorial Design of Experiments (DOE). The application involved four parameters (factors) with different possible values each. These parameters are given as follows:

Population size: 15, 30
Subgroups number: 2–9
Construction method: OPS, TPS, IP, EP
Iteration number: 250 or 25 min CPU time

The makespan value of the best schedule produced in each run is an indication of the parameters' combined performance.

The size of the $F_m(P_{m1},\ldots,P_{mm})/size_{ij}/C_{max}$ problem changes from 5 jobs × 2 stages to 100 jobs × 8 stages. Processing times were generated randomly from the range (1; 100). These problems are NP hard even for $k = 2$ [3]. This problem can be formally stated as follows: a set of n ($n = 5$, 10, 20, 50, and 100) jobs to be processed in a k ($k = 2$, 5, and 8) stage flow shop identical parallel processor. There are 240 instances. The *P50S8T05* problem is defined as 50 jobs, 8 stages, 05 problem indexes and *P* type problem. There are two types of problem; *P* and *Q*. In type *P* problems, the number of processors available at various stages is randomly selected from the set ($m_i = 2,\ldots,5$), whereas in type *Q* problems, it is fixed with $m_i = 5$ for all stages. The *P* and *Q* type problems are known as easy and hard problems, respectively. In the *P1HS8T05* problems the *HS* represents the 100 jobs. Totally 240 problems are classified into 24 groups according to their characteristics [2]. An instance prob-

Table 3
The performances of GA and PGA.

Problems	APD			
	GA better	PGA better	Equal	Total
P Type	31	**46**	43	120
Q Type	37	**57**	26	120
Total	68	**103**	69	240

lem is taken from each of the groups. Parameter optimization is implemented and best parameter sets are found for the instance. The parameter set found for an instance is generalized and used for other problems in the same group. Therefore, parameter optimization is implemented for 24 instances. The best parameters set of each benchmark problem is given in Table 1.

3. Computational results

The aim of the computational study is to analyze the performance of PGA to minimize the makespan for HFSMT problems. The results are compared with the earlier studies of Oğuz [2] and Oğuz and Ercan [3]. The solution files of Oğuz's problems were received from the authors to make a comparison [2]. The algorithm was implemented in Borland Delphi and ran on a PC Pentium 4 processor with 3 GHz and 512 MB memory.

The results obtained by using the proposed PGA and Oğuz's GA solutions are presented in Table 2. The results are presented in terms of Average Percentage Deviation (APD) of the solution from the Lower Bound (LB).

The LB of the optimal makespan was developed by Oğuz et al. [1] and defined by the following formula:

$$LB = \max_{i \in M} \left\{ \min_{j \in J} \left\{ \sum_{k=1}^{i-1} p_{kj} \right\} + \frac{1}{m_i} \sum_{j \in J} p_{ij} size_{ij} + \min_{j \in J} \left\{ \sum_{k=i+1}^{m} p_{kj} \right\} \right\} \quad (1)$$

where *J* is the set of jobs and *M* is the set of stages.

Also APD is defined as in the following:

$$APD = \frac{BestC_{\max} - LowerBound}{LowerBound} \times 100 \quad (2)$$

In Table 2, the column named as *problem* represents the name of the problem family; *GA* represents the APD results of Oğuz's [2] GAs; *PGA* represents the proposed parallel greedy algorithm with APD and CPU times measured in minutes. The better solution of the benchmark problems obtained by PGA is presented in bold letters.

As it is seen in Table 2, for *P* types problems, the proposed PGA found smaller APD values for 46 benchmark problems over 120 while GA found only 31 better solutions. Also, PGA and GA found equal APD values for 43 benchmark problems in all 120 instances.

For *Q* type problems, PGA found smaller APD values for 57 benchmark problems over 120 while GA found only 37 better solutions. Also, PGA and GA found equal APD values for 26 benchmark problems over 120 *Q* type instances.

When all problems are considered, PGA found smaller APD values for 103 benchmark problems over 240 while GA found only 68. Also, PGA and GA found equal APD values for 69 benchmark problems over 240 instances. These results are summarized in Table 3. The results shown in Table 3 are significantly better than the results obtained by Oğuz [2].

Average Percentage Values (APV) of the proposed PGA are also compared with the earlier GA study of Oğuz and Ercan [3] and TS study of Oğuz et al. [1] for *Q* type ($m_i = 5$) hybrid flow shop scheduling with multiprocessor task problems.

Table 4
Comparative APV results of TS, GA, and PGA.

Stage	Job	APV		
		TS[a]	GA[b]	PGA
2	10	4.61	4.61	13.50
2	20	2.63	1.84	9.23
2	50	2.12	1.48	4.63
2	100	2.13	1.49	1.83
Average		**2.87**	**2.36**	7.29
5	10	9.83	6.88	7.49
5	20	11.45	7.98	6.82
5	50	20.32	14.22	5.47
5	100	29.47	20.10	7.18
Average		17.76	12.30	6.74
8	10	18.51	12.96	14.72
8	20	22.88	17.67	16.21
8	50	27.58	19.30	14.71
8	100	32.42	22.70	11.76
Average		25.34	18.16	14.35
Total average		15.32	10.94	9.46

[a] TS of Oğuz et al. [1].
[b] Best GA solutions of Oğuz and Ercan [3].

The APV of PGA is defined as follows:

$$APV = \frac{\sum_{l=1}^{l} APD(L)}{l} \quad (3)$$

The results are presented in Table 4. In Table 4, the APV of PGA is compared with TS and GA.

As it is seen in Table 4 TS and GA have found the smaller average APV than PGA for 2 stage problems. For 5 stage problems, average APV for TS, GA and PGA are 17.76, 12.30, and 6.74, respectively. Also for 8 stage problems, average APV for TS, GA, and PGA are 25.34, 18.16, and 14.35, respectively. As it is noticed from average APV for 5 and 8 stage problems, PGA has found better results than TS and GA. If all *Q* type problems are considered; the total average APV of PGA is 9.46 while the total averages of TS and GA are 15.32, and 10.94, respectively. The improvement rates for total average APV using PGA are defined as follows:

$$\text{Improvement Rate}(\%) = \frac{(\text{Total average APV of PGA} - \text{Total average APV of TS or GA})}{\text{Total average APV of TS or GA}} \times 100 \quad (4)$$

On the total average APV of PGA, an improvement of 35.25% with respect to TS and 13.52% with respect to GA have been achieved.

The comparison of PGA, GA, and TS with respect to CPU times has no meaning since the configurations of the computers are different. We note that the maximum CPU time spent among all application data by the proposed PGA is 4.24 min.

Anyway, in Table 5, CPU times of PGA, GA, and TS are compared even if they come from different computer configurations.

Table 5
The maximum CPU times of the proposed PGA, GA and TS.

	Configuration of the computer	Limit of CPU time (min)
PGA	PC Pentium 4–3 GHz—512 memory	25
GA	PC Pentium 4–2 GHz—256 memory	30
TS	PC 686–800 MHz	266.44

4. Conclusion

In this paper, the $F_m(P_{m1}, \ldots, P_{mm})/size_{ij}/C_{max}$ problem which is NP-Hard in the strong sense is considered and a PGA is developed. The four constructive heuristic methods are proposed in order to be used in the developed PGA. The performance results of various population sizes, subgroups number, construction method and iteration number or stopping criteria for HFSMT problems are examined.

The proposed new approach is tested on a set of 240 instances taken from the literature. The computational results are compared with the earlier studies of Oğuz et al. [1], Oğuz [2] and Oğuz et al. [27]. The obtained results are encouraging since the proposed PGA gave the best solutions for 42.91% of the all *P* and *Q* type problems. And also for *Q* type problems the proposed PGA gave the smaller total average APV than Oğuz and Ercan's GA [3] and Oğuz et al.'s TS [1]. This is the first time that PGA is applied to $F_m(P_{m1}, \ldots, P_{mm})/size_{ij}/C_{max}$ problem. The computational results indicated that the proposed PGA approach is effective in terms of reduced makespan for the attempted problems. The proposed PGA is a good problem solving technique for a scheduling problem.

For future research, PGA heuristics may be used for some other industrial problems.

Acknowledgements

The authors would like to thank Ceyda Oğuz and Fikret Ercan for benchmark problems, solution files, and all the help they provided.

References

[1] C. Oğuz, Y. Zinder, V.H. Do, A. Janiak, M. Lichtenstein, Hybrid flow shop scheduling problems with multiprocessor task systems, European Journal of Operational Research 152 (2004) 115–131.
[2] C. Oğuz, Data for Hybrid Flow-shop Scheduling with Multiprocessor Tasks, Koç University, 2006, Available from http://home.ku.edu.tr/~coguz/.
[3] C. Oğuz, M.F. Ercan, A genetic algorithm for hybrid flow shop scheduling with multiprocessor tasks, Journal of Scheduling 8 (2005) 323–351.
[4] J.N.D. Gupta, Two stage hybrid flow shop scheduling problem, Journal of Operational Research Society 39 (4) (1988) 359–364.
[5] M.C. Portmann, A. Vignier, D. Dardilhac, D. Dezalay, Branch and Bound crossed with GA to solve hybrid flowshops, European Journal of Operational Research 107 (1998) 389–400.
[6] T.S. Arthanari, K.G. Ramamurthy, An extension of two machines sequencing problem, Opsearch 8 (1971) 10–22.
[7] S.A. Brah, J.L. Hunsucker, Branch and bound algorithm for flow shop with multiple processors, European Journal of operations Research 51 (1991) 88–99.
[8] J.A. Hoogeveen, J.K. Lenstra, B. Vettman, Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard, European Journal of Operational Research 89 (1996) 172–175.
[9] F. Riane, A. Artib, S.E. Elmaghraby, A Hybrid three stage flow shop problem: efficient heuristics to minimize makespan, European Journal of Operational Research 109 (1998) 321–329.
[10] N. Grangeon, A. Tanguy, N. Tchernev, Generic simulation model for hybrid flow shop, Computers and Industrial Engineering 37 (1999) 207–210.
[11] O. Moursli, Y. Pochet, A branch and bound algorithm for the hybrid flow shop, International Journal of Production Economics 64 (2000) 113–125.
[12] E.G. Negenman, Local search algorithms for multiprocessor flow shop scheduling problem, European Journal of Operational Research 128 (2001) 147–158.
[13] R. Linn, W. Zhang, Hybrid flow shop scheduling: a survey, Computers and Industrial Engineering 37 (1999) 57–61.
[14] O. Engin, A. Döyen, A new approach to solve hybrid flow shop scheduling problems by artificial immune system, Future Generation Computer Systems 20 (2004) 1083–1095.
[15] L. Tang, W. Liu, J. Liu, A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment, Journal of Intelligent Manufacturing 16 (2005) 361–370.
[16] M. Zandieh, S.M.T.F. Ghami, S.M.M. Husseini, An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times, Applied Mathematics and Computation 180 (2006) 111–127.
[17] H. Allaoui, A. Artiba, Scheduling two stage hybrid flow shop with availability constraints, Computers & Operations Research 33 (2006) 1399–1419.
[18] M. Haouari, L. Hidri, A. Gharbi, Optimal Scheduling of a two stage hybrid flow shop, Mathematical Methods of Operations Research 64 (2006) 107–124.
[19] P. Caricato, A. Grieco, D. Serino, TSP-based scheduling in a batch-wise hybrid flow shop, Robotics and Computer-Integrated Manufacturing 23 (2007) 234–241.
[20] A. Janiak, E. Kozan, M. Lichtenstein, C. Oğuz, Metaheuristic approaches to hybrid flow shop scheduling problem with a cost related criterion, International Journal Of Production Economics 105 (2007) 407–424.
[21] S. Vob, A. Witt, Hybrid flow shop scheduling as a multi-mode multi project scheduling problem with batching requirements: a real world application, International Journal of Production Economics 105 (2007) 445–458.
[22] K. Alaykıran, O. Engin, A. Döyen, Using ant colony optimization to solve hybrid flow shop scheduling problems, International Journal of Advanced Manufacturing Technology 35 (2007) 541–550.
[23] K.C. Ying, S.W. Lin, Multiprocessor task scheduling in multistage hybrid flowshops: an ant colony system approach, International Journal of Production Research 44 (16) (2006) 3161–3177.
[24] S.F.S. Şerifoğlu, G. Ulusoy, Multiprocessor task scheduling in multistage hybrid flow shops: a genetic algorithm approach, Journal of the Operational Research Society 55 (2004) 504–512.
[25] S.H.H. Edwin, N. Ansari, H. Ren, A genetic algorithm for multiprocessor scheduling, IEEE Transactions on Parallel and Distributed Systems 5 (2) (1994) 113–120.
[26] C. Oğuz, F.M. Ercan, Scheduling multiprocessor tasks in a two stage flow shop environment, Computers and Industrial Engineering 33 (1997) 269–272.
[27] C. Oğuz, M.F. Ercan, T.C.E. Cheng, Y.F. Fung, Heuristic algorithms for multiprocessor task scheduling in a two stage hybrid flow shop, European Journal of Operational Research 149 (2003) 390–403.
[28] S.F.S. Şerifoğlu, I.U. Tiryaki, Multiprocessor task scheduling in multistage hybrid flow-shops: a simulated annealing approach, Proceedings of the 2nd International Conference on Responsive Manufacturing (2002) 270–274.
[29] Y.K. Kwok, I. Ahmad, On multiprocessor task scheduling using efficient state space search approaches, Journal of Parallel and Distributed Computing 65 (2005) 1515–1532.
[30] Y. Zinder, V.H. Do, C. Oğuz, Computational complexity of some scheduling problems with multiprocessor tasks, Discrete Optimization 2 (2005) 391–408.
[31] P.E. Coll, C.C. Ribeiro, C.C. Souza, Multiprocessor scheduling under precendence constraints: polyhedral results, Discrete Applied Mathematics 154 (2006) 770–801.
[32] M.H. Shenassa, M. Mahmoodi, A novel intelligent method for task scheduling in multiprocessor systems using genetic algorithm, Journal of the Franklin Institute 343 (2006) 361–371.
[33] L. Kuszner, M. Malafiejski, A polynomial algorithm for some preemptive multiprocessor task scheduling problems, European Journal of Operational Research 176 (2007) 145–150.
[34] S.C. Cheng, D.F. Shiau, Y.M. Huang, Y.T. Lin, Dynamic hard-real time scheduling using genetic algorithm for multiprocessor task with resource and timing constraints, Expert Systems with applications 36 (2009) 852–860.
[35] R. Hwang, M. Gen, H. Katayama, A comparison of multiprocessor task scheduling algorithms with communication costs, Computers & Operations Research 35 (2008) 976–993.
[36] C.T. Tseng, C.J. Liao, A particle swarm optimization algorithm for hybrid flowshop scheduling with multiprocessor tasks, International Journal of Production Research 46 (17) (2008) 4655–4670.
[37] K.C. Ying, An iterated greedy heuristic for multistage hybrid flowshop scheduling problems with multiprocessor tasks, Journal of the Operational Research Society, doi: 10.1057/palgrave.jors.2602625.
[38] S. Binato, W. Hery, D. Loewenstern, M. Resende, A GRASP for job shop scheduling, in: C. Riberio, P. Hansen (Eds.), Essays and Surveys on Metaheuristic, Kluwer Academic Publishers, 2001, pp. 59–79.
[39] R.M. Aiex, S. Binato, M.G.C. Resende, Parallel GRASP with path-relinking for job shop scheduling, Parallel Computing 29 (2003) 393–430.
[40] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for permutation flowshop scheduling problem, European Journal of Operational Research 177 (2007) 2033–2049.
[41] D. Baraz, G. Mosheiov, A note on a greedy heuristic for flow shop makespan minimization with no machine idle-time, European Journal of Operational Research 184 (2008) 810–813.
[42] A. McAndrew, Greedy algorithms, http://sci.vu.edu.au/~amca/SCM2711/ch02_greedy.pdf (2005).
[43] D. Kowalski, Greedy algorithms: minimum spanning tree, http://www.csc.liv.ac.uk/~darek/COMP523/lecture7.pdf (2007).
[44] J. Shapiro, Search Methods part I, http://www.cs.man.ac.uk/~jls/CS2411/Search.pdf.gz (2006).
[45] M. Müller-Hannemann, K. Weihe, The greedy algorithm Section 3.2. http://www.algo.informatik.tu-darmstadt.de/ (2004).