MATEMATIKA, 2014, Volume 30, Number 1a, 71-84
©Department of Mathematical Sciences, UTM.

Online scheduling in Minimizing Makespan on Identical Parallel Processor with Release Date

[1]Syarifah Zyurina Nordin and [2]Lou Caccetta

[1]Department of Mathematical Science, Faculty of Science, Universiti Teknologi Malaysia, 81310 UTM Johor Bahru, Johor, Malaysia
[2]Western Australian Centre of Excellence in Industrial Optimization (WACEIO), Department of Mathematics and Statistics, Curtin University of Technology, GPO BOX U1987, Perth 6845, Western Australia, Australia
e-mail: [1]szyurina@utm.my, [2]L.Caccetta@exchange.curtin.edu.au

Abstract. In this paper, we address a non-preemptive task scheduling problem with an objective function of minimizing the makespan. We consider online scheduling with release date on identical parallel processing system with centralized and no splitting structure. Multi-steps heuristic algorithms are proposed to solve this non-deterministic scheduling problem. A computational experiment is conducted to examine the effectiveness of the proposed multi-steps method in different size problem. The computational results show that all the proposed heuristics obtain good results with the gap between optimal solutions are less than 10% even for a large date set. The experiment is performed using Microsoft Visual C++ programming software in windows environment.

Keywords Online scheduling; Minimizing makespan; Identical parallel processor; Release date.

2010 Mathematics Subject Classification 68M20, 90B36, 90C27.

1 Introduction

This paper studies the non-deterministic aspect in identical parallel processor systems. An added feature is considered to the standard task characteristic, which is the online scheduling with different task release dates. Online scheduling refers to the availability of task information. All the problem instances can only be known during the course of scheduling. Online scheduling is in contrast to offline scheduling where all the information of the tasks are ready and known before the execution starts. Heuristic algorithms are developed as the optimum solution for the problem can only be obtained for the offline model. Our proposed heuristics for online scheduling problem are based on the multi-step algorithm and will be applied in the task selection phase.

For a literature review survey about online scheduling, Sgall [1] presented a comparative results analysis for different types of online paradigms. He mentioned in one theorem that for any variant of online scheduling of job arrival over time, there exist a 2-competitive algorithm with respect to makespan. Lee et al. [2] provided a survey for online scheduling specifically in minimizing the makespan subject to machine eligibility. They proceed with the results for online over list and online over time. There are review focussed on the competitive analysis of the online algorithms by Albers [3].

Different algorithms have been developed for solving online scheduling. For example, Hurink and Paulus [4] showed the use of a greedy algorithm, where the online over list problems have a competitive ratio strictly not less than 2. Fu et al. [5] considered an online over time problem on unbounded batch machine to minimize makespan with limited restart. They provided an algorithm with a competitive ratio of 3/2. Tian et al. [6] proposed an online over time algorithm with a competitive ratio no greater than $\sqrt{2}$. They showed the bound is tight for the problem on two parallel batch machines with the objective of minimizing the makespan.

Some researchers also considered online scheduling with other task characteristics. For example, Epstein [7] addressed online scheduling with precedence constraints. The lower bound is proved to be $2-\frac{1}{m}$ on the competitive ratio of any deterministic algorithm and $2-\frac{2}{1-m}$ on any randomized algorithm. Both results are applied to ($1-m$) preemptive or non-preemptive cases. Huo et al. [8] also considered precedence constraints and also other features with a set of equal-processing time with the objective of minimizing the makespan.

Our work focuses on online scheduling with a different task release date in minimizing the makespan. This problem also has many applications in industry. For example, identical parallel machine scheduling systems can be found in real-world manufacturing environment such as in the integrated-circuit packing manufacturing particularly in wirebonding workstation [9]. The wirebonding operations also deal with online job arrival case and minimizing the makespan as the objective. Online scheduling on identical parallel processors also can be applied in another application involving high tech equipment on operating rooms in health care industry [10]. A utilization schedule needs to be prepared daily as well as monthly to fit the room with patients' cases (as many as possible) and facilities needed for each. The room scheduler will consider the operating room as the identical machine and the cases represent the jobs. The room utilization or the makespan is very critical to profitability.

2 Problem Description

The parallel processor system can be stated as follows: We are given a set of n tasks, J_i = J_1, J_2, …, J_n, that are to be processed on a set of m parallel processors, M_j = M_1, M_2, …, M_m. The tasks are independent in that there are no precedence relations among them. The processors are said to be identical, when the processing time $p_{ij} = p_i$ for all tasks i and processor j with the same processor speed. The performance criterion of interest is the makespan, C_{max} of the system. In real-time systems, it is important to minimize the makespan, which is the maximum of total tasks execution times for every processor, to balance the capacity utilization among parallel processors.

We are concerned with scheduling policies in parallel processing systems with centralized and no splitting system structure. In such systems, there are $m > 1$ identical processors. A set of sequence of tasks arrives in the system at random points in time with arbitrarily distributed inter-arrival times. When a task arrives into the system, it is immediately queued up in order and centralized (the queue is common for all the m processors). Each task requires an amount of random service time. The scheduling of tasks on the processors has no splitting structure since the entire set of tasks are scheduled to run sequentially on the same processor once the task are scheduled.

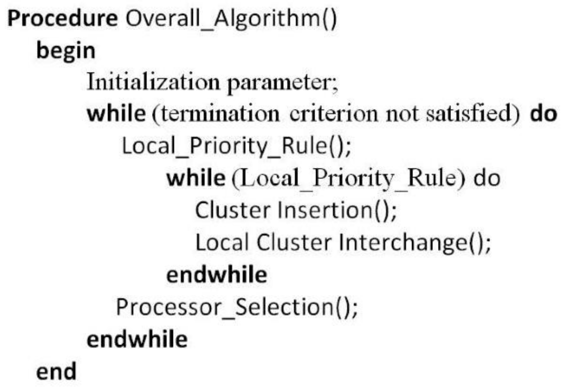
In this scheduling problem, we assume that all processors are available to serve at time zero and can process not more than one task at a time. Each processor is continuously available and there is no idle time between the execution of a pair of tasks. Each task has a processing requirement with positive processing time units. Each arrival tasks will be assigned to one processor only, inferring that the task migration between processors is prohibited. The processing of any operation may not be interrupted. Every task execution must be completely done before another task is assigned to the same processor and each task must be successfully assigned to only one processor. Moreover, each task i has a non-negative integer release date, r_i, at which time it becomes available for processing.

We consider the scheduling with online characteristic where all information is not known in advance. The scheduling decision by the scheduler has to be made at execution time and in this model, the operation is irrevocable. All previous decisions to assign and schedule a task also may not be revoked. In online scheduling, there are several classifications of possibilities on the arrival tasks. The most attention on online model systems has been focused on online over list and online over time scheduling. Online over list systems do not notice the existence of the arrival task until all its predecessors have already been scheduled while online over time systems know the existence of the tasks at their release date. In our problem, the way of task characteristics information received is the model of scheduling online over time. Note that, for the multiprocessor systems, there are no online scheduling algorithms that can be optimal in most cases [11]. It is due to the lack of global information on the instances [12]. The optimum value for online scheduling with task release dates can only be obtained when the scheduling is performed offline.

From all these descriptions and assumptions, our task scheduling problem can specifically be denoted based on the established three-filed notation as $P|\text{online},r_i|C_{max}$, i.e. the task scheduling problem for minimizing the makespan on identical parallel processors with the model requires scheduling to be online over time where the availability of each task is restricted by release date. We have added a substantial degree of difficulty to the classical problems $P||C_{max}$ which is already strongly NP-hard [13].

3 Proposed Heuristics Algorithm for Online Scheduling

In this section, we consider three heuristic algorithms that required less arrangement time for solving $P|\text{online},r_i|C_{max}$ which are simple and fast. The fast arrangement time is needed as for the online scheduling, we have to compete with the running time. The proposed algorithms that we are going to introduce share a common structure. First, task selection phase is obtained with a loop of multi-step procedures. The proposed heuristics design a slot for a refinement step during the running time in the loop and produce new Q_{list} for each refinement step. At the first step of the loop, we apply priority rule for the whole system until the solution is obtained. Along with this step, two subsequent modification methods are applied but still in the priority rule loop. This modification steps are based on a Cluster Insertion (CI) and a Local Cluster Interchange (LCI) methods. After obtaining a list of candidate in the task selection phase, the procedure is continued with processor selection phase until the system is terminated. The overall structure of the proposed heuristics is given in Figure 1.



```
Procedure Overall_Algorithm()
  begin
      Initialization parameter;
      while (termination criterion not satisfied) do
         Local_Priority_Rule();
            while (Local_Priority_Rule) do
               Cluster Insertion();
               Local Cluster Interchange();
            endwhile
         Processor_Selection();
      endwhile
  end
```

Figure 1: Pseudo-algorithm for proposed heuristic

3.1 Task Selection Scheme

3.1.1 Local Priority Rule

A priority rule in task scheduling is a policy in assigning an available task to an idle processor with a specific sequencing decision [14]. An available task refers to a waiting task that is ready to be assigned. The goal of applying a priority based rule is to select the task which has to be scheduled next, so that the system correctly allocates the resource. The rule also gives acceptable results with a reasonable computational time and is easy to implement [15]. The task selection is important to predict the outcome expressed by the objective function. The priority rules can be classified into two types [14]: local and global. The distinction between the local and global rules is in term of the information status rule. In local rule, the information on the waiting jobs at the machine is for a current time only. A global rule requires information on the jobs beyond the corresponding queue. The local information of the queue can be applied to our parallel processing system with centralized and no splitting structure. In local priority rule, the information is also more specific to the workstation. The local rules are widely used because they are robust against disruption [16]. Our local rule has time independent priority computation. Obviously, the processing time does not change over time once they have been computed in a queue.

In the parallel processing system, ready tasks share a single waiting list. In multi-step method, First-Come-First-Serve (FCFS) priority rule is computed at the first stage of the method and produces a local priority rule (LPR) scheduling list, Q_{LPR}. Figure 2 depicts the process of the LPR. FCFS is a random attribute in the local queue and a simple task priority rule which can often be found in real-systems but is usually considered inadequate by others. Note that, a FCFS discipline implies that services begin in the same order as arrival, but that tasks may have a different order because of different-length service time. The multi-step scheduler serves the unscheduled tasks in the waiting list using a specific task selection process that will be discussed in the next subsection.

```
Procedure Local_Priority_Rule()
  begin
    for $t = 1$ to $t = t_{\max}$ do
      if $t_1 \le t_2 \le t_{\max}$ do
        for $i = 1$ to $n$ do
          $Q_{list}$ := perform candidates for $Q_{LPR}$
        endfor
      endif
    endfor
  end
```

Figure 2: Pseudo-algorithm for Local Priority Rule

FCFS policies have been seen to be optimal in terms of the number of tasks and the throughput (i.e. the number of activities that run to completion within the given amount of time) [17]. There are cases where FCFS also exhibit a nice property regarding response time and smaller vectors of transient response time for all tasks. However, sometimes the optimality properties of FCFS fail to hold when tasks entering the system have such a random structure. The multi-step method can be effectively employed to improve the system. In order to escape from this randomization problem, we propose the following two modification procedures in the multi-step method loop: Cluster Insertion (CI) and Local Cluster Interchange (LCI).

3.1.2 Cluster Insertion (CI)

We define an insertion for the $P|\text{online},r_i|C_{max}$ problem in the tasks sequence where one task is moved from a current Q_{list} to an improved Q_{list} . The CI procedure is a build up phase for refinement purpose in the local priority rule loop. In this phase, the tasks are extracted from Q_{LPR} and inserted one by one to a build up phase that constructs a cluster. We employ this CI idea from clustering method in proposed heuristic algorithm in the literature for the general clustering problem. Most of them consider the clustering problem in offline scheduling. In more detail, clustering is a method of gathering tasks together and mapping them in the same group. Formally, the clustering method is either based on linear clustering or non-linear clustering. Linear clustering is a situation where the tasks gather in the same cluster and dependent to each other as they have precedence relations among them. The CI process can be considered as non-linear clustering. The step involves two or more tasks in a cluster where the tasks are independent. These independent tasks can be easily distributed to parallel processors since there is no interference among the processors and other tasks. Therefore, there is no restriction in partitioning the tasks in CI process. CI can provide some advantages when the task is moved into several partitions. The random structure of the arrival tasks in the system

```
Procedure Cluster_Insertion()
  begin
    Candidates $Q_{LPR}$;
    while (status=1) do
      for $t=1$ to $t=t_{\max}$ do
        $a_{i,t_a}$ := the arrival of task $i$ at time $t_a$;
        $r_{i,t_b}$ := the release date of task $i$ at time $t_b$;
        if ( $a_{i,t_a} < r_{i,t_b}$ ) then
          for $k=1$ to $k=\zeta$ do
            Extract candidate from $Q_{LPR}$ and assign to cluster;
            $Q_{list}$ := perform cadidates for $Q_{LI}$;
          endfor
        else
          Processor_Selection();
        endif
      endfor
```

Figure 3: Pseudo-algorithm for Cluster Insertion procedure

could be simply managed. Therefore, the randomization issue in FCFS local priority rule can be partly solved by CI and completely solved in the LCI phase that we will discuss in the next section.

Obviously, in order for the movement to be accepted, firstly, the task *i* has to be in the Q_{LPR} i.e $J_i \in Q_{LPR}$ $\mathrm{J_i} \in \mathrm{Q_{LPR}}$. Secondly, the release date of the task must be greater than the arrival time. Otherwise the mapping won't succeed. Notice that this system is continuous and idle time between two tasks is unaccepted. More specifically, for the $P|\text{online},r_i|C_{max}$ problem, the CI procedure consists of dividing tasks in Q_{LPR} and insert them into particular ζ slots. This insertion procedure is essentially repeated for *n* times. All mapped tasks are deleted from Q_{LPR} and inserted into a new list denoted as $Q_{CI.}$ Figure 3 shows a clear description of this procedure of CI.

3.1.3 Local Cluster Interchange (LCI)

The LCI method is a permutation process specifically employed in the multi-step method loop after CI is adopted. In this process, the clusters that have been constructed in Q_{CI} will have mutation process to produce a new Q_{list} named $Q_{LCI.}$ The mutation technique works by swapping the tasks within the local cluster in order to control the structure of the system. The task swapping process in cluster *A* will apply a simple and fast list scheduling (LS) algorithm to produce permutation cluster *A'*. This step is applied until all clusters are done. To initiate the permutation process, the first cluster in Q_{CI} is chosen and stored as ζ_1. Then, the procedure continues for the second cluster in the list. The second cluster is stored as ζ_2. This step is then applied for all clusters in Q_{CI}. Once this stage is accomplished, the local cluster interchange operation begins.

For a Q_{LCI} to be obtained, only one LS algorithm can be applied for the whole Q_{list}. In this proposed heuristic, we opt for the two well known LS algorithms for the interchange process. More specifically, the LS for the problem $P|\text{online},r_i|C_{max}$ are Longest Processing Time (LPT) and Shortest Processing Time (SPT) algorithms. For every cluster, we evaluate the processing time according to the selected LS method. Then, after the LS procedure, all the tasks in the cluster are ready in their own positions. If the task ahead is already assigned, the position is empty. The next task is able to replace and transfer to the empty position and the process continues for all remaining tasks in the Q_{LCI} .

An initial task for a cluster is chosen by LPT algorithm where the longest processing time among the task i.e $\mathrm{p_1} \geq \mathrm{p_2} \geq \cdots \geq \mathrm{p_i}$ is selected. The same procedure is repeated until the last cluster is reached. Similarly, the SPT also has to define the starting task and continue until the final task in a cluster but by using the shortest processing time approach where $\mathrm{p_1} \leq \mathrm{p_2} \leq \cdots \leq \mathrm{p_i}$ is applied. This procedure is to form Q_{LCI}. The owner of the first position in Q_{LCI} is the first task which will be picked up for the next procedure. The pseudo-algorithm for LCI procedure is shown in Figure 4.

```
Procedure Local_Cluster_Interchange()
  begin
    Candidates $Q_{CI}$;
    while (status=1) do
      for $\zeta = 1$ to $\zeta = \zeta_{\max}$ do
        store the cluster accordingly as $\zeta_1, \zeta_2, ..., \zeta_{\max}$;
        for $i = 1$ to $i = J_{\max}$ do
          for $k = 1$ to $k = \zeta$ do
           swap the tasks and sort according to the LS algorithm;
           $Q_{list}$ := perform cadidates for $Q_{LCI}$;
          endfor
        endfor
```

Figure 4: Pseudo-algorithm for Local Cluster Interchange procedure

3.2 Processor Selection Scheme

We now present a processor selection algorithm after multi-step loop in the task selection process. We have a list of candidates that are ready to get served. In online scheduling, the random processor selection is not the wisest approach. Therefore, the best way of choosing which processor to be assigned once a task has been selected is by the greedy way. We develop our heuristics with a simple and fast rule; and also can produce good results in this dynamic environment. The algorithm that we apply is a search method for the earliest idle processor to compose the selected task. The status of the processor will be updated for each time slot. During the tracking, if an available processor is found at the time slot, the status of the processor will be 0, otherwise 1. The selected task is accepted to be assigned to the first processor with status 0 only. However, there might be more that one processor with status 0 at same time *t*. In this case, the task can choose the processor with the smaller index. In brief, the algorithm can be stated as follows:

Step 1: Define the status of the processors. If the processor is busy, the status is 1 otherwise it is 0 for idle.

Step 2: At time $t = 0$, initialize the status of all processors with 0. This means that, all processors are idle and there are no tasks in the system.

Step 3: At time $t \geq 0$, check the status of the processor at each time slot from the lowest index to the highest index label. Select the first processor with status 0 and assign to the task. Discard other processors with status 1.

Step 4: If there is no processor available at that time slot, discard all processors and continue checking the status for the next time slot. Repeat the process for all processors at each time slot until all tasks have been assigned.

The final step of the heuristic is the status update phase. The status of the system must be updated to reflect the new changes. The system will stop when all tasks are assigned to the processors.

4 Computational Experiment

In this section we present a computational experiment on the proposed algorithm for solving the $P|\text{online},r_i|C_{max}$. The implementation is performed to evaluate the effectiveness of the proposed algorithm. For this purpose, we present three implementations of the algorithms and test their performances. We evaluate these heuristics and present the best heuristic. We also reveal the gaps with the optimum value from the MILP model obtained by Funk et al. [11]. The following are the three heuristics that we use in the computational testing:

HA 1: The HA 1 algorithm implements LPR, CI and LCI in the task selection process. In LCI, each of the $\zeta_1, \zeta_2, \ldots, \zeta_{\max}$ contains a random list scheduling with no specific sequence order to form $Q_{LCI.}$

HA 2: This algorithm puts together the multi-step method in the overall algorithm in Figure 1 with LPR, CI and LCI for the task selection phase. It has to be noted that in LCI, HA 2 applies LPT to form the formation in the interchange procedure. The final Q_{list} obtain by HA 2 is Q_{LCI} before they get transferred to processor selection stage.

HA 3: As HA1 and HA 2, HA 3 also carry out the procedure in this order: LPR, CI and LCI. The final Q_{list} of the task selection multi-step in HA 3 is Q_{LCI}. We also have to note that in LCI of HA 3, we implement SPT in the interchange procedure before proceed to the next level which is processor selection.

We implement our HA 1, HA 2 and HA 3 using Microsoft Visual C++ 6.0 on a personal computer with Intel Core 2 2.66 GHz 1.95 GB RAM. The algorithms are in a dynamic environment where the task information can only be known during the execution over time. We generate optimum solution using AIMMS 3.10 software package. The simulation data for the problem $P|\text{online},r_i|C_{max}$ is generated as follows:

1. The number of independent tasks are n={200, 400, 600, 800,1000}.
2. For every set of tasks, we have ζ={10, 20, 30, 40, 50}.
3. For every combination of n and ζ, we have $m = 3$ and $m = 5$.
4. The processing time for the instances is assumed to follow a discrete uniform distribution between 1 to 60 i.e. distribution U[1,60].
5. We generate 20 instances for every combination. Therefore, in this experiment, we have n x ζ x m = 5 x 5 x 2 = 50 combinations that produced 50 x 20 = 1000 instances.

4.1 Computational Results

We now present our performance results of the HA 1, HA 2 and HA 3 algorithms compared with the optimal solutions. We report the solutions for the objective function (i.e. makespan) for every instance, *I*. The average percentage deviation from the optimum, Gap_a, are examined and can be calculated as follows:

$$Gap_a(\%) = \frac{1}{20}\sum_{I=1}^{20}\frac{C_{\max}(I) - C*_{\max}(I)}{C*_{\max}(I)} \times 100$$

where $C_{\max}(I)$ is the makespan obtained by the heuristic for instance *I* and $C^*_{\max}(I)$ is the makespan of the optimum solution for instance *I*.

The maximum gap, Gap_w, of the instances for every combination was also observed using the following formulation:

$$Gap_w(\%) = \max\left\{\frac{C_{\max}(I) - C*_{\max}(I)}{C*_{\max}(I)} \times 100 \middle| I = 1,2,...,20\right\}$$

4.1.1 Performance of the best heuristics

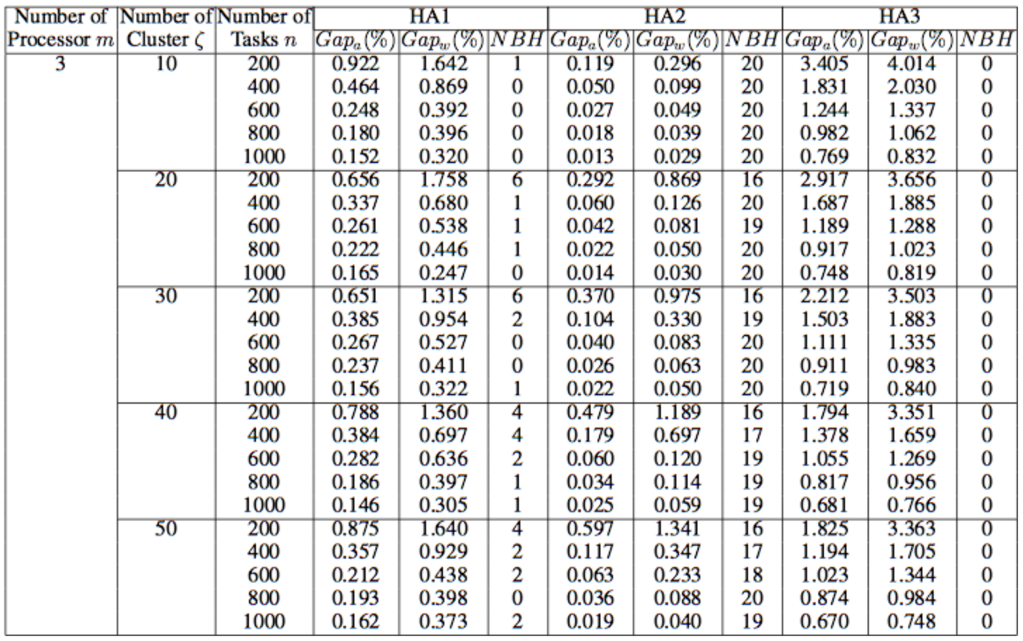
In this section, we present the computational results obtained from the computational experiment of HA 1, HA 2 and HA 3 algorithms. We will discuss the results delivered by the best heuristics. Tables 1 and 2 give for each heuristic algorithm the average value for the gap compared with the optimum solutions as a function of the problem size (m, ζ, n). We also evaluate the algorithms with each other by reporting the number of instances in which the heuristic becomes the best heuristic denoted as *NBH*. If there is more than one heuristic that obtained the same best solution for a certain instance, those particular algorithms can be declared as the best heuristic for that instance. We also reported the Gap_w for every combination from the 20 instances to observe the worse performance by the algorithms. It is observed that all instances in the problem shown in Tables 1 and 2 can be solved within 1 second on average.

The tables already give us clear observation that the algorithms of HA 1, HA 2 and HA 3 achieved a very good performance, where the average gap is less than 6.13% from the optimum for all size combinations. The best heuristic is HA 2, which is very outstanding, where the maximum Gap_a is only 1.099% for the case m = 5, ζ = 50 and n = 200. As can clearly be seen from the result in Table 1 and 2, HA 2 always obtain the lowest Gap_a for all different size of problems. From the *NBH*, HA 2 is the best heuristic with 95.4% from 1000 generated test problems. The HA 1 and HA 3 also produced good results with 2% and 6% of the maximum Gap_a respectively for the same case m = 5, ζ = 10 and n = 200.

5 Conclusion

In this paper, we introduced one of the additional features in parallel processing system that we explored in our study which is online scheduling with release date in minimizing the makespan. This feature is dynamic and therefore, we applied a multi-step method to reduce the non-determinism in the online scheduling. We partition the scheduling process into three phases: (1) local priority rule; (2) cluster insertion; and (3) local cluster interchange.

Table 1: Performance comparison of the heuristic algorithm with the optimum solution for m = 3



Number of Processor m	Number of Cluster ζ	Number of Tasks n	HA1			HA2			HA3		
			$Gap_a(\%)$	$Gap_w(\%)$	NBH	$Gap_a(\%)$	$Gap_w(\%)$	NBH	$Gap_a(\%)$	$Gap_w(\%)$	NBH
3	10	200	0.922	1.642	1	0.119	0.296	20	3.405	4.014	0
		400	0.464	0.869	0	0.050	0.099	20	1.831	2.030	0
		600	0.248	0.392	0	0.027	0.049	20	1.244	1.337	0
		800	0.180	0.396	0	0.018	0.039	20	0.982	1.062	0
		1000	0.152	0.320	0	0.013	0.029	20	0.769	0.832	0
	20	200	0.656	1.758	6	0.292	0.869	16	2.917	3.656	0
		400	0.337	0.680	1	0.060	0.126	20	1.687	1.885	0
		600	0.261	0.538	1	0.042	0.081	19	1.189	1.288	0
		800	0.222	0.446	1	0.022	0.050	20	0.917	1.023	0
		1000	0.165	0.247	0	0.014	0.030	20	0.748	0.819	0
	30	200	0.651	1.315	6	0.370	0.975	16	2.212	3.503	0
		400	0.385	0.954	2	0.104	0.330	19	1.503	1.883	0
		600	0.267	0.527	0	0.040	0.083	20	1.111	1.335	0
		800	0.237	0.411	0	0.026	0.063	20	0.911	0.983	0
		1000	0.156	0.322	1	0.022	0.050	20	0.719	0.840	0
	40	200	0.788	1.360	4	0.479	1.189	16	1.794	3.351	0
		400	0.384	0.697	4	0.179	0.697	17	1.378	1.659	0
		600	0.282	0.636	2	0.060	0.120	19	1.055	1.269	0
		800	0.186	0.397	1	0.034	0.114	19	0.817	0.956	0
		1000	0.146	0.305	1	0.025	0.059	19	0.681	0.766	0
	50	200	0.875	1.640	4	0.597	1.341	16	1.825	3.363	0
		400	0.357	0.929	2	0.117	0.347	17	1.194	1.705	0
		600	0.212	0.438	2	0.063	0.233	18	1.023	1.344	0
		800	0.193	0.398	0	0.036	0.088	20	0.874	0.984	0
		1000	0.162	0.373	2	0.019	0.040	19	0.670	0.748	0

Table 2: Performance comparison of the heuristic algorithm with the optimum solution for *m* = 5.

Number of Processor m	Number of Cluster ζ	Number of Tasks n	HA1			HA2			HA3		
			Gap_a(%)	Gap_w(%)	NBH	Gap_a(%)	Gap_w(%)	NBH	Gap_a(%)	Gap_w(%)	NBH
5	10	200	2.091	10.398	1	0.892	9.185	19	6.130	14.644	0
		400	0.878	1.373	0	0.107	0.292	20	3.065	3.354	0
		600	0.519	0.895	0	0.063	0.194	20	2.123	2.261	0
		800	0.441	0.895	0	0.037	0.105	20	1.643	1.813	0
		1000	0.429	1.728	1	0.099	1.529	19	0.591	2.825	0
	20	200	1.758	2.914	0	0.715	1.529	20	4.277	5.578	0
		400	0.985	1.689	0	0.194	0.444	20	2.651	3.224	0
		600	0.666	1.125	0	0.098	0.195	20	1.985	2.363	0
		800	0.380	0.631	1	0.042	0.100	20	1.537	1.774	0
		1000	0.273	0.558	1	0.040	0.082	20	1.256	1.336	0
	30	200	1.681	3.339	4	0.992	2.323	16	2.821	4.672	0
		400	0.930	0.930	1	0.206	0.759	20	2.448	3.133	0
		600	0.598	0.916	0	0.148	0.283	20	1.803	2.104	0
		800	0.455	0.747	1	0.073	0.165	20	1.418	1.646	0
		1000	0.274	0.589	0	0.045	0.100	20	1.132	1.336	0
	40	200	1.696	3.247	1	0.942	2.248	19	2.690	3.620	0
		400	0.904	1.376	1	0.425	1.262	19	2.122	2.851	0
		600	0.675	1.084	0	0.145	0.352	20	1.688	2.051	0
		800	0.398	0.772	1	0.097	0.298	20	1.305	1.693	0
		1000	0.322	0.694	0	0.061	0.181	20	1.137	1.308	0
	50	200	1.647	2.758	4	1.099	2.473	18	2.653	3.986	0
		400	0.709	1.319	7	0.520	1.401	14	1.642	2.613	0
		600	0.547	0.898	0	0.204	0.515	20	1.446	1.967	0
		800	0.414	0.804	1	0.094	0.210	20	1.322	1.711	0
		1000	0.358	0.587	0	0.060	0.149	20	1.107	1.350	0

The different phases in the multi-step are the improvement for the next step in the algorithm. We developed these algorithms by their simplicity and practical usage in real practice. All the three algorithms are very efficient and produced a very good quality result with very small average gap. HA 2 is reported as the best heuristic with the smallest average gap and always be the best heuristics with a large number of *NBH*.

Acknowledgements

The authors would like to thank MOHE, Universiti Teknologi Malaysia (UTM) and RUG vote number 08J11 for funding the project.

References

[1] Sgall, J. *On-line scheduling*. Berlin: Springer. 1998.

[2] Lee, K., Leung, J. Y. T. and Pinedo, M. L. On-line scheduling with machine eligibility. *A Quarterly Journal of Operations Research*. 2010. 8: 331–364.

[3] Albers, S. On-line algorithms: a survey. *Mathematical Programming*. 2003. 97: 3-

26.

[4] Hurink, J. and Paulus, J. Online scheduling of parallel jobs on two machines is 2-competitive. *Operations Research Letters*. 2008. 36: 51–56.

[5] Tian, J. F. R., Yuan, J. and He, C. On-line scheduling on a batch machine to minimize makespan with limited restarts. *Operations Research Letters*. 2008. 36: 255–258.

[6] Tian, J. F. R. and Yuan, J. A best online algorithm for scheduling on two parallel batch machines. *Theoretical Computer Science*. 2009. 410: 2291–2294.

[7] Epstein, L. A note on-line scheduling with precedence constraints on identical machines. *Information Processing Letters*. 2000. 76: 149–153.

[8] Huo, Y., Leuong, J. Y. T. and Wang, X. On-line scheduling of equal-processing time task systems. *Theoretical Computer Science*. 2008. 401: 85–95.

[9] Yang, T. An evolutionary simulation-optimization approach in solving parallel-machine scheduling problems – a case study. *Computers & Industrial Engineering*. 2009. 56: 1126-1136.

[10] Vairaktarakis, G. L. and Cai, X. The value of processing flexibility in multipurpose machines, *IIE Transactions.* 2003. 35: 763-774.

[11]Funk, S., Goossens, J. and Baruah, S. On-line scheduling on uniform multiprocessor. *22nd IEEE Proceeding of Real-time Systems Symposium*. 2001. 183-192.

[12] Tao, J., Chao, Z. and Xi, Y. A semi-online algorithm and its competitive analysis for a single machine scheduling problem with bounded processing times. *Journal of Industrial and Management Optimization*. 2010. 6: 269-282.

[13] Du, D. Z. and Pardalos, P. *Handbook of Combinatorial Optimization*. Kluwer Academic Publishers. 1998.

[14] Haupt, R. A survey of priority rule-based scheduling. *OR Spektrum*. 1989. 11: 3-16.

[15] Klein, R. Bidirectional planning: Improving priority rule-based heuristic for scheduling resource-constraint projects. *European Journal of Operations Research*. 2000. 127: 619-638.

[16] Hartmann, W., Fischer, A. and Nyhuis, P. The impact of priority rules on logistic objectives: modeling with logistic operating curve. *Proceedings of the World Congress on Engineering and Computer Science*. 2004. 2.

[17] Baccelli, F., Liu, Z. and Towsley, D. Extremal scheduling of parallel processing with and without real-time constraints. *Journal of Association for Computing Machinery*. 1993. 40: 1209-1237.