MINIMIZING THE NORMALIZED SUM OF SQUARE FOR WORKLOAD DEVIATIONS ON IDENTICAL PARALLEL MACHINES

Johnny C. Ho, D. Abbott Turner College of Business, Columbus State University, Columbus, GA 31907
Tzu-Liang (Bill) Tseng, Dept of Mechanical & Industrial Engr., U. of Texas El Paso, El Paso, TX 79968
Alex J. Ruiz-Torres, Dept of Information and Decision Sciences, U. of Texas El Paso, El Paso, TX 79968
Francisco J. López, School of Business, Macon State College, Macon, GA 31206

ABSTRACT

In many organizations, it is desirable to distribute workload as equally as possible among a group of employees or machines. This paper proposes a performance measure, that we call the Normalized Sum of Square for Workload Deviations (*NSSWD*), and studies the problem of how to schedule a set of *n* jobs on *m* parallel identical processors in order to minimize the *NSSWD*. The *NSSWD* criterion is relevant where uniformity of wear to machines or of workload to employees is desirable. An algorithm, called Workload Balancing (WB), is proposed for solving this problem. Moreover, we perform a simulation experiment to evaluate WB against several well-known heuristics in the literature. Lastly, we discuss the computational results obtained from the simulation experiment.

1. INTRODUCTION

Efficient utilization of resources is critical to the operations of any organization and scheduling plays an important role in achieving this goal. One key characteristic of efficient resource utilization is the balancing of work across the production resources given that it allows all resources to be 'spent' equally, and eliminates problems caused when one resource is assigned more work than another.

This paper addresses the problem of scheduling *n* jobs, each job *j* with process time p_j, in *m* identical parallel machines with the objective of balancing the work load across machines as evenly as possible. Let C_i be the completion time of the last job in machine *i* and $C_{max} = \mathbf{max.}_{\,i=1\ldots m}\,[C_i]$, which is also known as the makespan of the schedule. Let μ be the mean machine completion time (the average completion time for the last job in all machines), $\mu = 1/m \cdot \sum_{i=1 \ldots m} C_i$. Note that the value of μ is also equal to $1/m \cdot \sum_{i=1 \ldots n} p_j$, which is a constant given a problem instance. A "balanced work load" schedule will be one where all C_i's are as close to μ as possible.

Figure 1 presents four schedules for the same set of jobs. There are four machines (e.g. $m = 4$) and eight jobs (e.g. $n = 8$) with process times {7, 5, 4, 4, 3, 3, 3, 3}. The mean machine completion time is 8 time units and shown in the schedules as a dashed line. Schedules 1 and 2 have a makespan of 10, but we propose that schedule 2 has a more balanced loading, thus from a schedulers standpoint, schedule 2 will be preferred to schedule 1. Both schedules 3 and 4 have a makespan of 9 time units, which is the best possible makespan solution. It is also proposed that schedule 4 is better than schedule 3 given that the loading is more balanced (only 2 machines are off the target μ in schedule 4, versus all machines off in schedule 3). From a load balancing point of view, schedule 4 is the best possible schedule.

The literature only has one previously proposed criterion used to measure the load balancing performance of a parallel machines schedule. This method, proposed by Rajakumar et al. (2004), is called Relative Percentage of Imbalance (*RPI*) and is based on the average error relative to C_{max}. It is defined as $RPI = 1/m \cdot \sum_{i=1 \ldots m} (C_{max} - C_i) / C_{max}$. The problem with using this criterion in the case of identical parallel machines is that this measure depends solely on C_{max}, and does not assess workload balancing as shown next:

$$1/m \cdot \sum_{i=1 \dots m} (C_{max} - C_i) / C_{max} = 1/(m\, C_{max}) \cdot \sum_{i=1 \dots m} (C_{max} - C_i)$$
$$= 1 - 1/(m\, C_{max}) \cdot \sum_{i=1 \dots m} C_i$$
$$= 1 - \mu / C_{max}.$$

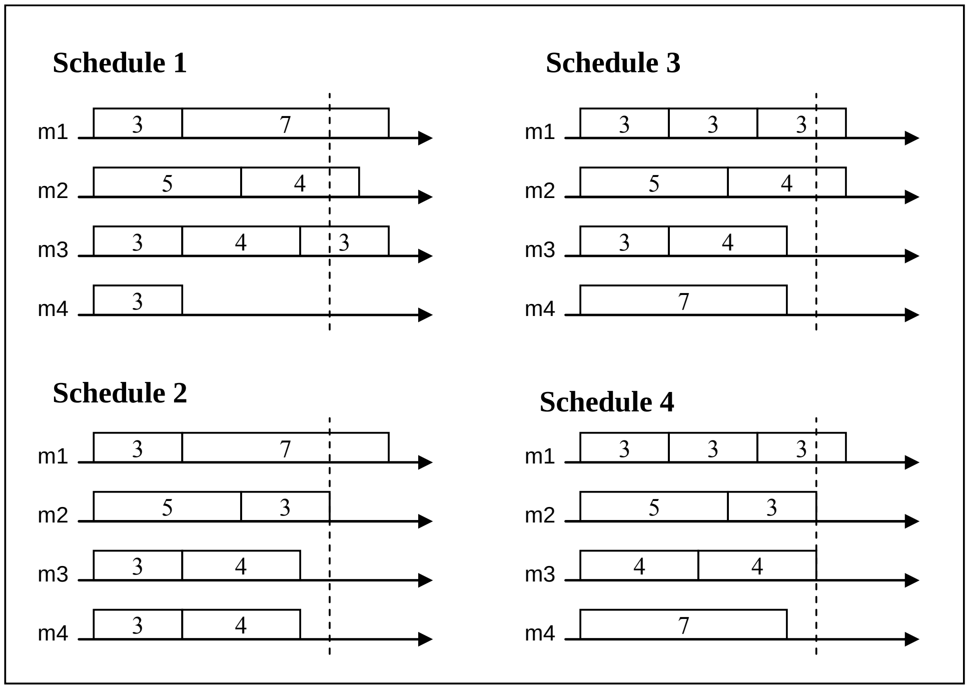

FIGURE 1: EXAMPLE OF THE PROBLEM

Thus, the RPI measure converts to $1 - \mu / C_{max}$, so for the case of the identical parallel machine problem, the RPI criterion is just a transformation of the C_{max} criterion. Despite this, the RPI could be an applicable measure when the machines are not identical, case in which $\sum_{i=1 \dots m} C_i$ may not be a constant. The RPI measure for schedules 1-4 in Figure 1 are 0.2, 0.2, 0.111, and 0.111 respectively.

This paper proposes a new work load balancing measure based on the square errors that can be used in the case of identical parallel machines. Square error measures have been used for many problems, for example in variance determination, regression analysis, forecast errors, and design of experiment (Montgomery, 2005). A measure based on square errors is relevant as it "penalizes" both above and below target results, and the size of the penalty increases quadratically, thus heavily penalizing large deviations. This paper proposes the use of a normalized sum of squared errors measure as a way to characterize the load balancing of a schedule. The concept of the proposed measure is based on the well-established *Sum of Squares Principle* in measuring variability in statistics and serves as a precise and quantifiable measurement criterion.

For a single machine i the square error is given by $(C_i - \mu)^2$, with a total for all machines provided by $\sum_{i=1 \dots m} (C_i - \mu)^2$. Different problem instances usually have distinct characteristics like different number of machines or jobs. Thus, in order to support the relative evaluation of multiple problem instances, a normalized square root of this sum is used in this paper as the measure of load balanced. We propose the Normalized Sum of Square for Workload Deviations (*NSSWD*) criterion as a performance measure that can characterize the load balancing across parallel resources; defined as $NSSWD = [\sum_{i=1 \dots m} (C_i - \mu)^2]^{½} / \mu$. Clearly the problem's objective is the minimization of *NSSWD*. For the schedules presented in Figure 1, the *NSSWD* measure values are 0.728, 0.306, 0.25, and 0.177, respectively for Schedules 1 to 4.

The *NSSWD* problem is a special case of the “assembly-line balancing” problem (Conway et al., 1967) and carries a wide spectrum of general applications in several disciplines, including manufacturing, logistics, and computer science (for example those mentioned in Brown 1971, Ho and Chang, 1991, Dyckhoff and Finke, 1992, Khouja and Conrad, 1995). Examples of relevance today include manual assembly cells often found in the *maquila* industry where it is highly desirable to assign work to parallel production cells in such a way that all work is completed by a common time and that no cell is overloaded and requires overtime. An ideal schedule would assign jobs such that all production cells are assigned the same amount of work and all finish it at the same time. Besides job processing applications, the parallel machine problem is similar to the bin packing problem, thus an application of interest would be the loading of trucks for example, and load balancing would relate to the cargo assigned to each of the tucks in the fleet. Having all trucks assigned equal loads would maximize the overall use of the resources and keep all drivers equally satisfied.

As mentioned earlier, the identical parallel machines load balancing problem is related to the identical parallel machine makespan problem. For comprehensive reviews on parallel machines scheduling research, refer to Cheng and Sin (1990), Lam and Xing (1997), and Mokotoff (2001). A significant amount of research on parallel machines scheduling has been directed towards solving the makespan problem (Graham, 1969, Coffman et al., 1978, Lee and Massey, 1988, Ho and Wong, 1995, Gupta and Ruiz-Torres, 2001, Lee et al., 2006, Akyol and Bayhan, 2006). Although a makespan optimal schedule could be *NSSWD* optimal, as in the case of Schedule 4 in Figure 1, there is no guarantee that this always occurs. For example, Schedule 3 is optimal for the makespan criteria, but not optimal for *NSSWD*. We propose that the likelihood that a C_{max} optimal schedule will not necessarily be a *NSSWD* optimal schedule gets larger as the number of machines increases. Despite this, an optimal or near optimal makespan solution is a good candidate to be close to an optimal or near optimal *NSSWD* solution.

Well-known heuristics for parallel machines scheduling makespan problems include: the Longest Processing Time (LPT) algorithm (Graham, 1969), the Multifit algorithm (Coffman et al., 1978), and the Repetitive Modified Greedy (RMG) algorithm (Lee and Massey, 1988). The LPT algorithm assigns jobs one by one, starting from the unscheduled job with the longest processing time, to the machine having the smallest assigned processing time. The Multifit algorithm uses a bin-packing approach along with a binary search to find the minimum capacity, that is, the minimum makespan. The RMG algorithm modifies the Multifit algorithm in one key respect: replacing the ‘first fit decreasing’ (FFD) criterion by the ‘repetitive modified greedy’ criterion. The Two-Machine Optimal (TMO) algorithm (Ho and Wong, 1995) utilizes the concept of lexicographic search to minimize the makespan for the two-machine case. Ho and Wong (1995) empirically show that TMO outperforms both Multifit and RMG and generally dominates Multifit and RMG in both makespan and CPU time measures. They also develop the extended TMO algorithm, denoted by xTMO, for the *m* parallel machines makespan problem. Recent uses of LPT based heuristics for parallel machine problems include Lin and Liao (2008) and Koulamas and Kyparsis (2008).

The remainder of the paper is organized as follows. The next section describes properties of the *NSSWD* measure and establishes the complexity of the problem. Section 3 discusses the proposed algorithm for the *NSSWD m* identical parallel processors problem. The fourth section provides a numerical example to illustrate the proposed algorithm. Section 5 presents the design of our simulation experiment which is used to evaluate the effectiveness of the proposed algorithm. Section 6 discusses the computational results obtained from the simulation experiment. Lastly, the seventh section concludes the paper.

2. PROPERTIES OF *NSSWD* AND PROBLEM COMPLEXITY

This section demonstrates some properties of the *NSSWD* measure and establishes the complexity of P||*NSSWD*.

Proposition 1. In the case of 2 machines, a C_{max} optimal schedule is also an *NSSWD* optimal schedule.

Proof. Let C_{max} and C_k be the completion times for the C_{max} optimal schedule S, where $C_{max} \geq C_k$. Let $C_x = C_{max} + \Delta$ and $C_z = C_k - \Delta$ be the machine completion times of another schedule S' not C_{max} optimal, with $\Delta > 0$ and all completion times also > 0. Then, $NSSWD(S) = [\,(C_{max} - \mu)^2 + (C_k - \mu)^2]^{\frac{1}{2}} / \mu$, and $NSSWD(S') = [\,(C_x - \mu)^2 + (C_z - \mu)^2]^{\frac{1}{2}} / \mu$. Suppose that $NSSWD(S) > NSSWD(S')$. Then $(C_{max} - \mu)^2 + (C_k - \mu)^2 > (C_x - \mu)^2 + (C_z - \mu)^2$, which simplifies to $C_{max}^{\ 2} + C_k^2 > C_x^{\ 2} + C_z^2$. By substitution, we obtain $C_{max}^{\ 2} + C_k^2 > (C_{max} + \Delta)^2 + (C_k - \Delta)^2 = C_{max}^{\ 2} + C_k^2 > C_{max}^{\ 2} + 2\Delta\, C_{max} + \Delta^2 + C_k^2 - 2\Delta C_k + \Delta^2$, which leads to $0 > 2\Delta\,(C_{max} + \Delta - C_k\,)$. This is a contradiction given that $C_{max} \geq C_k$ and $\Delta > 0$ by assumption. □

Proposition 2. In the case of $m > 2$ machines, a C_{max} optimal schedule is not necessarily a *NSSWD* optimal schedule.

Proof. Example: Consider a set of 5 jobs with processing times 10, 100, 50, 40, and 10 time units, respectively, to be processed by three identical machines. Let S be a schedule with $C_1 = 100$ (job 2 in m_1), $C_2 = 90$ (jobs 3 and 4 in m_2), and $C_3 = 20$ (jobs 1 and 5 in m_3). Note that S is makespan optimum. Let S' be the schedule that results from assigning jobs 1 and 2 to m_1; job 3 to m_2; and jobs 4 and 5 to m_3. The completion times of S' are $C'_1 = 110$ and $C'_2 = C'_3 = 50$. The *NSSWD* value for S is 0.8806 whereas the one for S' is 0.6999. Clearly the makespan optimum schedule S is not *NSSWD* optimum (S' is not *NSSWD* optimum either: just move job 1 to m_2 or m_3 to improve the measure, but the point of this proof is that just because a schedule is makespan optimum, it does not mean it is necessarily *NSSWD* optimum. Schedule 3 in Figure 1 is an example involving four machines). □

Proposition 3. A non C_{max} optimal schedule can be improved in terms of *NSSWD* by a reduction in its maximum machine completion time.

Proof. Let S be a non C_{max} optimal schedule with machine completion times $C_1, \ldots, C_y, \ldots. C_m$, where $C_1 > C_y \geq C_m$. Let a modified version of the schedule, S', have completion times $C'_1, \ldots, C'_y, \ldots, C_m$ where $C'_1 = C_1 - \Delta$, $C'_y = C_y + \Delta$, and $0 < \Delta < C_1 - C_y$ (otherwise the makespan increases). We propose that $NSSWD(S) > NSSWD(S')$. Suppose otherwise. Then $C_1^{\ 2} + C_y^{\ 2} \leq C'^{\ 2}_1 + C'^{\ 2}_y$. Hence $C_1^{\ 2} + C_y^{\ 2} \leq C_1^{\ 2} + C_y^{\ 2} + 2\Delta C_y - 2\Delta C_1 + 2\Delta^2$, which reduces to $C_1 - C_y \leq \Delta$. This contradict that $C_1 - C_y > \Delta$. □

Proposition 4. A *NSSWD* optimal schedule is necessarily a C_{max} optimal schedule.

Proof. Comes directly from the proof of Proposition 4. □

As shown by Bruno et al. (1974) and Garey and Johnson (1979), the makespan parallel-machine problem is known to be NP-hard. From the above propositions it is concluded that finding an optimal C_{max} schedule will be a component of finding an *NSSWD* optimal schedule. Since the P||C_{max} problem is NP-Hard, the P||*NSSWD* problem is NP-Hard.

3. ALGORITHM WORKLOAD BALANCING

The proposed algorithm, called Workload Balancing (WB), is based on some existing algorithms that minimize makespan because the *NSSWD* and the makespan criteria correlate positively. The WB algorithm consists of two major modules: (1) develop an initial solution by any existing heuristic; and (2) improve the current solution by creating a series of two-machine sub-problems and solving them via Ho and Wong's TMO algorithm (1995). In module 1, we suggest to employ a quick and simple heuristic, such as LPT, Multifit, or RMG, to obtain a seed solution. In module 2, the TMO algorithm employs a lexicographic search approach to determine an optimal makespan schedule for the two-machine problem. While TMO's worst-case complexity is exponential, it has been shown to be capable of finding an optimal solution quickly.

The following is a summary of notation that will be used in the presentation of the proposed algorithm.

n	the number of jobs
p_j	processing time of job j
m	the number of machines
m_i	machine i
M	the set of m machines
δ	the set of machines of the sub-problem which yields no makespan reduction
σ	the set of unavailable (evaluated) machines
C_i	the completion time of the last scheduled job on machine i

Without loss of generality, processing times are assumed to be integer. The WB algorithm is given below.

WB Algorithm

```
Inputs: $n$, $m$, $p_i$, $\delta = \sigma = \phi$, an existing heuristic for use in Step 1.

Step 1: If $m = 2$, then
apply TMO to obtain a current schedule for the problem and go to Step 6;
        else,
    use an existing heuristic to obtain a seed schedule.

Step 2: Find $\min_{i \in M}\{C_i\}$ and $\max_{j \in M}\{C_j\}$.

Step 3: If $(C_j - C_i) \le 1$, then
    go to Step 6;
        else,
    apply TMO to the sub-problem consisting of jobs assigned to $m_i$ and $m_j$.

Step 4: If TMO yields a makespan reduction for the sub-problem in Step 3, then
    update the current schedule, set $\sigma = \phi$ and return to Step 2;
        else,
    move machines $i$ and $j$ to $\delta$,
    find $\min_{\substack{a \in M \\ a \notin \delta,\sigma}}\{C_a \ge C_i\}$ and $\max_{\substack{b \in M \\ b \notin \delta,\sigma}}\{C_b \le C_j\}$,
    if both machines $a$ and $b$ do not exist, then
go to Step 6.

Step 5: If $(C_j - C_a) \ge (C_b - C_i)$, then
    update $i = a$;
else,
    update $j = b$.
        Return to Step 3.

Step 6: Stop. The current schedule is the best schedule identified by WB.
```

For m = 2, Step 1 of the WB algorithm applies TMO to develop a makespan optimal schedule for the problem, then goes to Step 6 for termination. For m > 2, Step 1 utilizes a selected existing heuristic to obtain a seed schedule for the problem. Step 2 locates the smallest load machine (m_i) and the largest load machine (m_j). Machines i and j form a sub-problem to be considered in the next step because they represent the best potential to additionally reduce workload deviation.

In Step 3, if it is infeasible to reduce makespan by re-scheduling jobs in m_i and m_j, i.e., $(C_j - C_i) \leq 1$, then WB goes to Step 6 for termination; otherwise, WB creates a sub-problem consisting of jobs assigned to m_i and m_j and applies TMO to the sub-problem. In Step 4, if a reduction on *NSSWD* is achieved in the previous step, then WB returns to Step 2 to restart this process; otherwise, WB locates machines a and b such that $\min_{\substack{a \in M \\ a \notin \delta, \sigma}} \{C_a \geq C_i\}$ and $\max_{\substack{b \in M \\ b \notin \delta, \sigma}} \{C_b \leq C_j\}$, respectively (note that it is possible that a = b if there is only one available machine remaining). If both machines a and b do not exist implying that further improvement in makespan is infeasible, then WB goes to Step 6 for termination.

Step 5 selects the two machines yielding the greatest potential in reducing workload deviation as follows: if $(C_j - C_a) \geq (C_b - C_i)$, then WB creates a sub-problem consisting of jobs assigned to m_a and m_j; otherwise, WB creates a sub-problem consisting of jobs assigned to m_i and m_b. The WB algorithm now restarts Step 3 to test the termination condition. Finally, Step 6 terminates the WB algorithm and outputs the current best schedule.

For the three or more parallel machines problem, we will use the notation LPT+, Multifit+, and RMG+ to identify the WB algorithm that employs initial or seed solutions from LPT, Multifit, or RMG, respectively. LPT+, Multifit+, and RMG+ are designed with the specific *NSSWD* optimization criterion in mind. We also need to bear in mind that LPT, Multifit, and RMG, and the heuristics that result from adapting the extended TMO algorithm (Ho and Wong, 1995) to them, denoted by xLPT, xMultifit, and xRMG, respectively, are designed to obtain optimal or near optimal makespan solutions. Proposition 2 establishes that a makespan optimal schedule is not necessarily *NSSWD* optimal, but by Proposition 3 an *NSSWD* optimal schedule must be makespan optimal. Because of this, it is not clear how good are the schedules obtained with heuristics LPT, Multifit, RMG, xLPT, xMultifit, and xRMG in terms of the *NSSWD* optimization criterion. In this article we test and compare, with computational experiments, how effective are makespan optimization heuristics in generating good *NSSWD* solutions, and, simultaneously we analyze whether LPT+, Multifit+, and RMG+ are significantly better than makespan optimization heuristics. Before we discuss our computational results, we present a small example next.

4. NUMERICAL EXAMPLE

A small numerical example with twelve jobs and four machines is used to demonstrate the WB algorithm. The processing times of the twelve jobs are generated from a discrete uniform distribution between 1 and 100. They are then sorted in non-increasing order for convenience and re-numbered as 1, 2, ..., 12. Their processing times are: 88, 84, 81, 79, 79, 69, 65, 56, 52, 41, 29, and 14. The WB algorithm is applied to the example using seed solutions obtained by the LPT, Multifit, and RMG heuristics. Nonetheless, we only focus on Multifit as the seed solution here for illustrative purpose.

Let S_i be the set of jobs assigned to m_i, then the seed solution obtained by Multifit is: S_1 = {1, 2, 12} with C_1 = 186; S_2 = {3, 8, 9} with C_2 = 189; S_3 = {4, 6, 10} with C_3 = 189; and S_4 = {5, 7, 11} with C_4 = 173. The *NSSWD* for the seed Multifit schedule obtained from Step 1 is 0.07175. Step 2 finds that the machines with the largest and smallest loads are 2 and 4, respectively. Since $C_2 - C_4 = 16$ which is >

1, the six jobs assigned to m_2 and m_4, i.e., jobs 3, 5, 7, 8, 9, and 11, form the first sub-problem which is solved by the TMO algorithm in Step 3. Since the TMO algorithm returns a smaller makespan for the sub-problem, the current schedule is updated to: S_1 = {1, 2, 12} with C_1 = 186; S_2 = {5, 8, 9} with C_2 = 187; S_3 = {4, 6, 10} with C_3 = 189; and S_4 = {3, 7, 11} with C_4 = 175. The *NSSWD* for the updated current schedule is reduced to 0.05914.

As an improvement is made, WB returns to Step 2 and seeks the new largest and smallest load processors, which are m_3 and m_4, to form the next sub-problem. Applying the TMO algorithm to the six jobs assigned to m_3 and m_4, WB obtains the following schedule: S_1 = {1, 2, 12} with C_1 = 186; S_2 = {5, 8, 9} with C_2 = 187; S_3 = {4, 7, 10} with C_3 = 185; and S_4 = {3, 6, 11} with C_4 = 179. The *NSSWD* for this new current schedule is 0.03379. Since an improvement is accomplished, the WB again returns to Step 2 and identifies that the largest and smallest load processors, which are m_2 and m_4, to form the next sub-problem. However, no improvement is made from solving the sub-problem consisting of jobs assigned to these two machines, WB places m_2 and m_4 in set δ and finds that a = 3 and b = 1 in Step 4. In Step 5, since $(C_2 - C_3) < (C_1 - C_4)$, WB updates j = 1 and returns to Step 3.

TABLE 1: PERFORMANCE RESULTS FOR THE EXAMPLE

Heuristic	*NSSWD*	$C_{\max}$
LPT	0.10136	196
xLPT	0.03010	187
LPT+	0.02084	187
Multifit	0.07175	189
xMultifit	0.03379	187
Multifit+	0.02084	187
RMG	0.03942	187
xRMG	0.03942	187
RMG+	0.02589	187

The WB algorithm terminates and yields a final schedule as: S_1 = {1, 3, 12}; S_2 = {5, 8, 9}; S_3 = {4, 7, 10}; and S_4 = {2, 6, 11} with *NSSWD* = 14.75. Table 1 presents the *NSSWD* and makespan results for the LPT, xLPT, LPT+, Multifit, xMultifit, Multifit+, RMG, xRMG, and RMG+ methods.

5. SIMULATION EXPERIMENT

A simulation experiment is performed to test the effectiveness of the proposed algorithm. For the two-processor case, we compare the WB algorithm with three of the most common heuristics for the makespan problem in the literature, namely LPT, Multifit, and RMG. For the *m*-processor case, we also include the extended TMO algorithm (Ho and Wong, 1995) in the evaluation. The number of iterations is set at 10 for both Multifit and RMG so as to make a fair comparison. It should be noted that $(1/2)^{10} \leq$ 0.1%.

The simulation study is divided into two phases. Phase 1 deals with the two processors case only; while phase 2 considers the three or more processors case. In phase 1, we study the effects of two factors; namely number of jobs and variance of processing times, have on LPT, Multifit, RMG, and WB. The numbers of jobs are set at ten levels: 7, 8, 9, 10, 11, 12, 25, 50, 75, and 100. These levels cover both odd and even numbers of jobs. The processing times are assumed to follow a discrete uniform distribution, *DU*(1, *b*), where *b* is set at three levels: 100, 300, and 500. For more details on data generation see Gupta et al. (2004). These two factors together make 30 problem sets. One hundred replications are performed for each set, which provides a total of 3,000 problems. As we demonstrate in Proposition 1a, WB determines the optimal *NSSWD* for the two machines case. Thus, WB will serve as the standard of comparison for heuristics and provide some insights into the performance of other heuristics.

Phase 2 evaluates nine heuristics – LPT, xLPT, LPT+, Multifit, xMultifit, Multifit+, RMG, xRMG, and RMG+ and considers two factors – number of jobs and number of machines. For $m = 3$ and 4, n is set at four levels: 13, 14, 15, and 16; for $m = 5$, 8, 11, and 14, the n/m ratio is set at three levels: 5, 7, and 9. Hence, a total of 20 combinations of n and m values are studied. The processing time is assumed to follow a discrete uniform distribution with parameters 1 and 300. Again, we run 100 replications for each set of problems; this provides a total of 2,000 problems. All methods are implemented in FORTRAN running on a Pentium 4-based microcomputer.

6. COMPUTATIONAL RESULTS

The performance results for the two-processor case are shown in Table 2. Each value in the Table represents the mean *NSSWD* of 100 replications for the respective set as specified by *n* and *b*. It should be noted that *NSSWD* is expressed in the percentage form in our computational results. Table 2 confirms that the WB algorithm yields the best performance among all methods as it is optimal. Among the other three heuristics being considered, RMG yields the best overall performance; while LPT has the worst overall performance. The number of jobs correlates negatively with *NSSWD* for all methods. With respect to processing time variability (determined by *b*), it does not have a major impact on the performance of the three heuristics; but it correlates negatively with *NSSWD* for WB. Moreover, it is interesting to note that while RMG performs significantly better than the LPT and Multifit when *n* is small (7 − 25), but the opposite holds when *n* is large (50 − 100).

TABLE 2: TWO-PROCESSOR *NSSWD* (IN %) RESULTS

	LPT			Multifit			RMG			WB		
n \ *b*	100	300	500	100	300	500	100	300	500	100	300	500
7	4.1283	4.1541	4.1468	2.9681	3.1931	3.2504	1.6093	1.6273	1.6369	1.2917	1.3334	1.3374
8	2.5684	2.5670	2.5894	2.1336	2.2624	2.2758	1.2294	1.2327	1.2473	0.6236	0.5741	0.5922
9	3.0682	3.0801	3.0658	1.9573	1.9831	1.9979	0.9213	0.8515	0.8630	0.2957	0.2448	0.2352
10	1.9213	1.9724	1.9938	1.4175	1.4245	1.4446	0.7873	0.7847	0.7978	0.2006	0.1609	0.1661
11	1.6644	1.6615	1.6640	1.3350	1.3431	1.3617	0.6667	0.7144	0.7481	0.1343	0.0844	0.0728
12	1.1929	1.2086	1.2137	0.8061	0.7907	0.7958	0.5277	0.5120	0.5398	0.1406	0.0480	0.0444
25	0.3988	0.3943	0.3821	0.2018	0.2149	0.2286	0.1258	0.1643	0.1713	0.0665	0.0181	0.0098
50	0.0710	0.0720	0.0712	0.0533	0.0511	0.0611	0.0873	0.1192	0.1265	0.0285	0.0109	0.0057
75	0.0447	0.0442	0.0449	0.0307	0.0307	0.0342	0.0943	0.1261	0.1310	0.0167	0.0064	0.0038
100	0.0225	0.0205	0.0196	0.0180	0.0121	0.0159	0.1113	0.1286	0.1328	0.0124	0.0036	0.0036

Table 3 gives the CPU time in seconds for each of the 30 experimental conditions of the two-processor case. As shown in the Table, LPT requires the least CPU time; while RMG needs the most CPU time. The CPU time for both Multifit and RMG increases sharply as *n* increases, but it only increases modestly for LPT and WB.

Table 4 presents the *m*-processor *NSSWD* computational results for each of the 20 experimental conditions. Among the three heuristics yielding an initial solution, RMG outperforms Multifit in every one of the 20 experimental conditions, which in turn outperforms LPT in all 20 experimental conditions.

The xTMO algorithm is able to significantly reduce *NSSWD* using seed solutions obtained by LPT, Multifit, and RMG in all 20 experimental conditions. Furthermore, Multifit+ (abbreviated as MF+ in Tables 4 and 5) outperforms xMultifit (abbreviated as xMF in Tables 4 and 5) in all 20 experimental conditions; RMG+ outperforms xRMG in 19 experimental conditions and ties in one; and LPT+ outperforms xLPT in 16 experimental conditions and ties in four. For a particular level of *m*, *NSSWD* decreases as the ratio of *n* to *m* increases. This observation is true for all nine heuristics evaluated.

TABLE 3: TWO-PROCESSOR CPU TIME (IN SECONDS) RESULTS

	LPT			Multifit			RMG			WB		
n \ *b*	100	300	500	100	300	500	100	300	500	100	300	500
7	0.00	0.00	0.00	0.05	0.00	0.05	0.15	0.05	0.05	0.05	0.10	0.00
8	0.05	0.05	0.00	0.15	0.05	0.00	0.10	0.15	0.10	0.00	0.05	0.05
9	0.05	0.00	0.00	0.15	0.00	0.00	0.15	0.05	0.10	0.05	0.10	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00	0.25	0.15	0.15	0.05	0.05	0.05
11	0.00	0.00	0.05	0.05	0.00	0.10	0.10	0.05	0.10	0.15	0.10	0.15
12	0.05	0.05	0.05	0.05	0.00	0.05	0.20	0.10	0.15	0.00	0.10	0.15
25	0.05	0.00	0.00	0.10	0.05	0.05	0.60	0.20	0.20	0.05	0.10	0.05
50	0.10	0.00	0.10	0.30	0.25	0.35	0.65	0.50	0.50	0.15	0.05	0.15
75	0.30	0.15	0.10	0.80	0.50	0.40	0.80	0.85	0.90	0.30	0.15	0.05
100	0.25	0.20	0.20	1.25	1.15	0.50	1.30	1.10	1.20	0.25	0.30	0.30
Mean	0.09	0.05	0.05	0.29	0.20	0.15	0.43	0.32	0.35	0.11	0.11	0.10

TABLE 4: *M*-PROCESSOR *NSSWD* (IN %) RESULTS

m	*n*	LPT	xLPT	LPT+	MF	xMF	MF+	RMG	xRMG	RMG+
3	13	3.2516	0.4251	0.3357	2.3893	0.6062	0.3355	1.4988	0.5660	0.3344
	14	3.0063	0.2944	0.2120	1.8822	0.3459	0.2034	1.1619	0.2634	0.2194
	15	2.5738	0.1387	0.1349	1.7936	0.1937	0.1312	0.9401	0.1838	0.1339
	16	2.2386	0.1125	0.0975	1.4093	0.1428	0.1058	0.8439	0.1360	0.0936
4	13	5.5444	1.9586	1.7424	4.7195	2.3271	1.7554	3.6574	2.6174	1.6620
	14	5.2124	1.6109	1.3872	4.2358	1.4455	1.2859	2.5161	1.5362	1.2662
	15	5.6640	1.0352	0.8196	3.4261	1.1045	0.8121	2.0887	1.1606	0.8075
	16	4.1880	0.6317	0.4654	3.0299	0.7449	0.5036	1.6077	0.6745	0.4354
5	25	3.3543	0.1822	0.1539	1.8016	0.2814	0.1562	1.1276	0.2728	0.1892
	35	1.6319	0.0811	0.0800	1.0630	0.0986	0.0871	0.5579	0.0892	0.0820
	45	1.0605	0.0634	0.0634	0.5901	0.0640	0.0634	0.3652	0.0634	0.0634
8	40	3.7126	0.2252	0.1848	2.0846	0.3311	0.2175	1.2792	0.3262	0.2168
	56	2.1023	0.1081	0.1044	1.0522	0.1266	0.1151	0.6830	0.1328	0.1115
	72	1.2640	0.0810	0.0810	0.6378	0.0844	0.0835	0.5678	0.0862	0.0823
11	55	4.3466	0.2522	0.2055	2.0037	0.3781	0.2417	1.3416	0.3964	0.2615
	77	2.2242	0.1191	0.1173	1.1386	0.1669	0.1272	0.6685	0.1536	0.1274
	99	1.4189	0.0865	0.0865	0.7376	0.0962	0.0874	0.5984	0.0930	0.0877
14	70	4.6954	0.2589	0.2182	2.0011	0.4054	0.3024	1.4598	0.4314	0.2672
	98	2.4338	0.1356	0.1340	1.2768	0.1734	0.1473	0.8507	0.1782	0.1391
	126	1.5249	0.1056	0.1056	0.9434	0.1154	0.1061	0.6359	0.1285	0.1056

Table 5 gives the CPU time in seconds for each of the 20 experimental conditions of the *m*-processor case. Similar to the results obtained in the two-processor case, LPT requires the least CPU time while

RMG requires the most CPU time among the three heuristics generating initial solutions. The additional CPU time required by the xTMO and WB algorithms depends on the particular heuristic providing the initial solution. xRMG and RMG+ take about 50% longer CPU time than that of RMG; on the other hand, xLPT and LPT+ take about 400% more CPU time than that of LPT. For all heuristics, the CPU time generally increases as the *n*/*m* ratio increases. It is interesting to note that LPT+ in many experimental conditions is faster than Multifit and RMG but delivers much lower *NSSWD*.

TABLE 5: *m*-PROCESSOR CPU TIME (IN SECONDS) RESULTS

m	*n*	LPT	xLPT	LPT+	MF	xMF	MF+	RMG	xRMG	RMG+
3	13	0.00	0.35	0.35	0.05	0.30	0.50	0.30	0.45	0.70
	14	0.00	0.20	0.20	0.05	0.10	0.40	0.25	0.40	0.45
	15	0.05	0.20	0.15	0.05	0.25	0.35	0.15	0.25	0.25
	16	0.00	0.15	0.30	0.10	0.30	0.25	0.25	0.25	0.35
4	13	0.00	0.00	0.15	0.10	0.35	0.20	0.10	0.15	0.20
	14	0.00	0.10	0.15	0.15	0.30	0.35	0.05	0.25	0.20
	15	0.05	0.15	0.20	0.10	0.35	0.35	0.20	0.25	0.35
	16	0.00	0.20	0.10	0.15	0.40	0.45	0.15	0.15	0.35
5	25	0.00	0.25	0.35	0.05	0.30	0.55	0.35	0.75	0.95
	35	0.10	0.35	0.35	0.35	0.70	0.65	0.40	0.70	0.75
	45	0.05	0.30	0.30	0.30	0.70	0.65	0.70	0.85	0.95
8	40	0.05	0.35	0.75	0.20	0.80	0.85	1.05	1.60	1.60
	56	0.20	0.55	0.45	0.40	1.10	1.20	1.50	2.05	1.95
	72	0.10	0.45	0.50	0.65	1.25	1.15	1.90	2.35	2.40
11	55	0.15	0.75	1.45	0.65	1.80	1.95	1.40	1.95	1.85
	77	0.10	0.90	0.70	0.50	2.05	2.05	2.05	2.60	3.40
	99	0.20	0.40	0.95	0.85	3.17	3.12	3.00	4.35	4.10
14	70	0.25	1.25	1.60	0.75	1.65	2.45	2.75	3.65	3.85
	98	0.35	1.25	0.90	1.25	3.10	3.60	3.20	4.80	5.50
	126	0.40	0.95	1.30	1.45	3.92	4.51	5.15	7.20	7.42

7. CONCLUSIONS

We propose an algorithm, called WB, to minimize the sum of square for workload deviations on *m* parallel identical machines. The WB algorithm is based on the idea from the TMO algorithm (Ho and Wong, 1995) and guarantees an optimal solution when the number of machines is two. We perform an extensive simulation study to test the effectiveness of WB against several existing algorithms, including LPT, Multifit, and RMG, and the extended TMO algorithm. The computational results show that both the WB and extended algorithms outperform the LPT, Multifit, and RMG significantly. Moreover, the WB algorithm, on average, reduces the *NSSWD* given by the extended algorithm by about 25%, while it takes about 15% more CPU time than that of the extended algorithm.

There exist a number of avenues worthy of future investigations. First, it is interesting to study the bi-criteria scheduling involving the *NSSWD* and another performance measure, such as total weighted flowtime. Second, it is useful to investigate the tradeoffs between *NSSWD* and a due-date related criterion, such as the number of tardy jobs, and analyze the Pareto (efficient) solutions. That is, determining how much improvement in the number of tardy jobs can be gained by allowing *NSSWD* to increase by one or more units. Third, it would seem logical and desirable to extend the proposed algorithm to solve the *NSSWD* non-identical parallel machines problem.

REFERENCES

[1] Akyol, D.E., and Bayhan, G.M. “Minimizing Makespan on Identical Parallel Machines using Neural Networks.” *Lecture Notes in Computer Science*, 2006, 4234, 553–562.

[2] Brown, A.R. *Optimum Packing and Depletion: The Computer in Space- and Resource-Usage Problem.* Amsterdam: Elsevier Publishing Company, 1971.

[3] Bruno, J., Coffman, E.G., and Sethi, R. “Scheduling Independent Tasks to Reduce Mean Finishing Time.” *Communications of the ACM*, 1974, 17, 382–387.

[4] Cheng, T.C.E., and Sin, C.C.S. “A State-of-the-Art Review of Parallel-Machine Scheduling Research.” *European Journal of Operational Research*, 1990, 47, 271–292.

[5] Coffman, E.G., Garey, M.R., and Johnson, D.S. “An Application of Bin-Packing to Multiprocessor Scheduling.” *SIAM Journal of Computing*, 1978, 7, 1–17.

[6] Conway, R. W., Maxwell, W. L., and Miller, L. W. *Theory of Scheduling.* Reading, MA: Addison-Wesley, 1967.

[7] Dyckhoff. H., and Finke, U. *Cutting and Packing in Production and Distribution: A Typology*. New York: Springer-Verlag, 1992.

[8] Garey, M.R., and Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: Freeman, 1979.

[9] Graham, R.L. “Boundaries on Multiprocessing Timing Anomalies.” *SIAM Journal of Applied Mathematics*, 1969, 17, 416–429.

[10] Gupta, J.N.D., Ho, J.C., and Ruiz-Torres, A.J. “Makespan Minimization on Identical Parallel Machines subject to Minimum Total Flowtime.” *Journal of Chinese Institute of Industrial Engineers*, 2004, 21, 220–229.

[11] Gupta, J.N.D., and Ruiz-Torres, A.J. “A LISTFIT Heuristic for Minimizing Makespan on Identical Parallel Machines.” *Production Planning and Control*, 2001, 12, 28–36.

[12] Ho, J.C., and Chang, Y.-L. “Heuristics for Minimizing Mean Tardiness for *m* Parallel Machines.” *Naval Research Logistics*, 1991, 38, 367–381.

[13] Ho, J.C., and Wong, J.S. “Makespan Minimization for *m* Parallel Processors.” *Naval Research Logistics*, 1995, 42, 935–948.

[14] Khouja, M., and Conrad, R. “Balancing the Assignment of Customer Groups among Employees.” *International Journal of Operations and Production Management*, 1995, 15, 76–85.

[15] Koulamas, C., and Kyparsis, G. “A Modified LPT Algorithm for the Two Uniform Parallel Machine Makespan Minimization Problem.” *European Journal of Operational Research*, *In Press*.

[16] Lam, K., and Xing, W.X. “New Trends in Parallel Machine Scheduling.” *International Journal of Operations and Production Management*, 1997, 17, 326–338.

[17] Lee, C.Y., and Massey, J.D. “Multiprocessor Scheduling: An Extension of the Multifit Algorithm.” *Journal of Manufacturing Systems*, 1988, 7, 25–32.

[18] Lee, W.-C., Wu, C.-C., and Chen, P. “A Simulated Annealing Approach to Makespan Minimization on Identical Parallel Machines.” *International Journal of Advanced Manufacturing Technology*, 2006, 31, 328–334.

[19] Lin, C-H., and Liao, C-J. “Makespan Minimization for Multiple Uniform Machines.” *Computers and Industrial Engineering*, 2008, 54, 983–992.

[20] Mokotoff, E. “Parallel Machine Scheduling Problems: A Survey.” *Asia-Pacific Journal of Operational Research*, 2001, 18, 193–242.

[21] Montgomery, D.C. *Design and Analysis of Experiments*, 6th edition. Hoboeken, NJ: John Wiley & Sons, 2005.

[22] Rajakumar, S., Arunachalam, V.P., and Selladurai, V. “Workflow Balancing Strategies in Parallel Machine Scheduling.” *International Journal of Advanced Manufacturing Technology*, 2004, 23, 366–374.