An Ant Colony Optimization Algorithm for Load Balancing in Parallel Machines with Sequence Dependent Setup Times

Timur Keskinturk
Department of Quantitative Methods, Faculty of Business Administration, Istanbul University

Mehmet B. Yildirim
Wichita State University, bayram.yildirim@wichita.edu

Mehmet Barut
Barton School of Business, Wichita State University

AN ANT COLONY OPTIMIZATION ALGORITHM FOR LOAD BALANCING IN PARALLEL MACHINES WITH SEQUENCE-DEPENDENT SETUP TIMES

Timur Keskinturk,[†] Mehmet B. Yildirim,[⊥][1]
Mehmet Barut[¥]

[†] Department of Quantitative Methods, Faculty of Business Administration
Istanbul University
Istanbul 34850, Turkey

[⊥] Department of Industrial and Manufacturing Engineering
Wichita State University
Wichita, KS 67260, USA

[¥] Department of Finance, Real Estate, and Decision Sciences
Barton School of Business
Wichita State University
Wichita, KS 67260, USA

Abstract: This study introduces the problem of minimizing average relative percentage of imbalance (ARPI) with sequence-dependent setup times in a parallel-machine environment. A mathematical model that minimizes ARPI is proposed. Some heuristics, and two metaheuristics, an ant colony optimization algorithm and a genetic algorithm are developed and tested on various random data. The proposed ant colony optimization method outperforms heuristics and genetic algorithm. On the other hand, heuristics using the cumulative processing time obtain better results than heuristics using setup avoidance and a hybrid rule in assignment.

Keywords: Load Balancing, Parallel-machine Scheduling, Sequence-Dependent Setups, Ant Colony Optimization, Genetic Algorithm, Heuristics

[1] Corresponding Author, Email: Bayram.yildirim@wichita.edu, Phone: +1-316-978 3426, Fax: +1-316-978 3742

AN ANT COLONY OPTIMIZATION ALGORITHM FOR LOAD BALANCING IN PARALLEL MACHINES WITH SEQUENCE-DEPENDENT SETUP TIMES

Timur Keskinturk,[†] Mehmet B. Yildirim,[⊥][2]
Mehmet Barut[¥]

[†] Department of Quantitative Methods, Faculty of Business Administration
Istanbul University
Istanbul 34850, Turkey

[⊥] Department of Industrial and Manufacturing Engineering
Wichita State University
Wichita, KS 67260, USA

[¥] Department of Finance, Real Estate, and Decision Sciences
Barton School of Business
Wichita State University
Wichita, KS 67260, USA

Abstract: This study introduces the problem of minimizing average relative percentage of imbalance (ARPI) with sequence-dependent setup times in a parallel-machine environment. A mathematical model that minimizes ARPI is proposed. Some heuristics, and two metaheuristics, an ant colony optimization algorithm and a genetic algorithm are developed and tested on various random data. The proposed ant colony optimization method outperforms heuristics and genetic algorithm. On the other hand, heuristics using the cumulative processing time obtain better results than heuristics using setup avoidance and a hybrid rule in assignment.

Keywords: Load Balancing, Parallel-machine Scheduling, Sequence-Dependent Setups, Ant Colony Optimization, Genetic Algorithm, Heuristics

[1] Corresponding Author, Email: Bayram.yildirim@wichita.edu, Phone: +1-316-978 3426, Fax: +1-316-978 3742

1. INTRODUCTION

This paper presents a mathematical model for a parallel-machine problem with sequence-dependent setups where the goal is to minimize total relative imbalance. Rajakumar et al. (2004, 2006, and 2007) observed that the minimization of imbalance may reduce idle time and work in process, maximize throughput, minimize the finished goods inventory, and lower operating expenses. Furthermore, the impact of the machine with the highest workload, representing a bottleneck that prevents achieving high system throughput, may be reduced by utilizing all machines as equally as possible.

Workload balancing has several applications. Aubry et al. (2008) pointed out the applications of workload balancing in the semiconductor manufacturing industry. Duman et al. (2008) showed how a balanced schedule can improve productivity in the manufacturing of aluminum on parallel continuous casting lines. Yildirim et al. (2007) provided examples from the service industry (such as nurse scheduling) and production planning in a machine shop. Hillier and Brandeau (2001) illustrate another application for workload balancing in printed circuit board assembly. Assigning tasks to workers in an office or shop floor is another viable example. To maintain the morale of the workforce, it is of utmost important to keep the assignment of jobs to employees working in the same area as equally balanced as possible (i.e., assign approximately equal workload) to minimize tension on the office or shop floor.

The goal of workload balancing is to distribute jobs/tasks to resources in such a way that the relative imbalance is minimized, and the utilization of resources is approximately equal. Rajakumar et al. (2004, 2006, and 2007) defined workload balancing as the minimization of total relative imbalance (the ratio of the difference between maximum completion time on all machines and individual completion time on a given machine, and the maximum completion time). Duman et al. (2008) and Yildirim et al. (2007) achieved workload balance by ensuring that the completion times on individual machines are within a certain percentage of each other while minimizing the total completion time on all machines. Aubry et al. (2008) achieved load balancing by having the same completion time on each machine.

Workload balancing on parallel machines has been studied in various environments. For example, Rajakumar et al. (2004, 2006, and 2007) considered the case where jobs have deterministic processing times on identical parallel machines. Furthermore, Rajakumar et al. (2007) analyzed the impact of having precedence constraints on workload balancing. Aubry et al. (2008) studied the problem on parallel multi-purpose machines with machine-dependent setups. Yildirim et al. (2007) and Duman et al. (2008) investigated workload balancing on unrelated parallel machines in the presence of sequence-dependent setups. At all settings and environments, the workload-balancing problem is closely related to difficult-operation research problems, such as the set-partitioning problem (Rajakumar et al., 2004, 2006, and 2007; Aubry et al., 2008) or vehicle-routing problems (Yildirim et al., 2007; Duman et al., 2008), which are strongly NP-hard problems. The researchers of this paper studied the workload-balancing problem using sequence-dependent setups, with the objective of minimizing total relative imbalance. In other words, the goal was to assign jobs to different machines and determine the sequence of jobs to minimize average relative percentage imbalance.

Chen and Powell (2003) observed that solving a problem for non-identical parallel machines for any objective is more complex than for identical parallel machines, and the sequence-dependent setup time will further complicate the problem. Allahverdi et al. (1999) provided a review of scheduling problems with setups. Fowler et al. (2003) analyzed parallel-machine problems having makespan, total weighted completion time, and total weighted tardiness objectives with sequence-dependent setups. Due to the complexity involved in terms of formulation and computational time, several methods have been proposed to solve parallel-machine problems with sequence-dependent setups: dynamic programming (Gascon and Leachman, 1998), branch and bound (Dietrich and Escudero, 1989), and heuristics (Pinedo, 1995). Tamaki et al. (1993) proposed a genetic algorithm to solve unrelated parallel-machine scheduling problems with resource constraints. Yalaoui and Chu (2003) and Tahar et al. (2006) proposed a linear programming approach for minimization of completion time in the presence of sequence dependent setup times and job splitting. Chen (2006) used heuristics and simulated annealing for unrelated parallel machines with a mean tardiness objective as well as secondary resource constraints and setups.

The contributions of this paper can be summarized as follows: (1) first, we present a mathematical model to formulate the load balancing of a parallel scheduling problem with sequence-dependent setups when the objective is minimization of total relative imbalance; (2) then, we propose an ant colony optimization metaheuristic to solve the resulting problem; and (3) finally, we analyze the performance of this heuristic under various conditions and compare its performance with nine simple dispatching rules (very similar to those used in the load-balancing literature) and a genetic algorithm. We also present results on the performance of the simple dispatching rules.

The organization of this paper is as follows: In the next section (Section 2), we present the notation utilized in this paper and propose a mixed-integer mathematical model that minimizes total relative imbalance. Section 3 and 4 present an ant colony optimization algorithm and a genetic algorithm for obtaining good results in a reasonable amount of time.

Section 5 presents heuristics to solve this problem. The experimental setup is followed by computational experimentation.

2. PROBLEM DEFINITION

In the parallel-machine scheduling problem with workload balancing with sequence-dependent setups, all problem parameters are assumed to be deterministic. K is the set of parallel machines, and $|K|$ denotes the number of machines, i.e., the cardinality of set K. Similarly, J is the set of jobs that needs to be processed on these machines. K_j is the subset of machines on which job j can be processed, and J_k is the subset of jobs that can be processed on machine k. Jobs are available at time zero. If job i precedes job j on machine k, then there is a sequence-dependent setup time of S_{ijk} between jobs i and j. Let p_{ik} be the processing time of job i on machine k. No preemption of jobs is allowed. Each job is processed on only one machine and only once.

In this problem, to maximize potential throughput, it is assumed that all feasible schedules are non-delay schedules. In other words, no machine is kept idle other than during the required sequence-dependent setups when there is an operation available for processing, so there is no forced idleness.

Let C_k be the total completion time of all jobs assigned to machine k. The imbalance is defined as $C_{\max} - C_k$, where the minimization objective will force $C_{\max}$ to have the value of maximum completion time of jobs on all parallel machines, i.e.,

$$C_{\max} = \max_{k \in K} C_k .$$

The relative imbalance on machine k is the ratio of imbalance and the maximum completion time on all machines, i.e.,

$$\text{relative imbalance}_k = \frac{C_{\max} - C_k}{C_{\max}}.$$

Below is a mathematical program to minimize average relative percentage imbalance with sequence-dependent setup times in an unrelated parallel machine. The goal of the mathematical model for parallel-machine workload balancing with sequence-dependent setup (PMWBSDS) is to schedule jobs on unrelated parallel machines to minimize the average relative percentage of imbalance (ARPI).

$$\min \left(\frac{1}{|K|} \sum_{k \in K} \frac{C_{\max} - C_k}{C_{\max}} \right) * 100 \tag{1}$$

Mathematically, the total completion time on machine *k*, C_k is defined as

$$C_k = \sum_{i \in J} y_{ik} p_{ik} + \sum_{i \in J_k} \sum_{j \in J_k} x_{ijk} s_{ijk} \qquad k \in K \tag{2}$$

where

$$y_{ik} = \begin{cases} 1 & \text{if job } i \text{ is assigned to machine } k \\ 0 & \text{otherwise} \end{cases}$$

and

$$x_{ijk} = \begin{cases} 1 & \text{if job } i \text{ is the immediate predecessor of job } j \text{ on machine } k \\ 0 & \text{otherwise} \end{cases}$$

Constraint (3) ensures that the maximum workload is greater than or equivalent to individual workloads.

$$C_{\max} \geq C_k \qquad k \in K \tag{3}$$

Constraint (4) ensures that each job is assigned to a processing line.

$$\sum_{k \in K_i} y_{ik} = 1 \qquad i \in J \tag{4}$$

Constraint (5) guarantees that a job cannot precede another job on machine *k* unless it has been assigned to machine *k*.

$$x_{ijk} \le y_{ik} \qquad i \in J, k \in K_i, j \in J_k \tag{5}$$

Constraint (6)/constraint (7) ensures that a job must be before/after another job on a production line.

$$\sum_{i \in J_k} x_{ijk} \le y_{ik} \qquad k \in K, j \in J_k \tag{6}$$

$$\sum_{j \in J_k} x_{ijk} \le y_{ik} \qquad k \in K, i \in J_k \tag{7}$$

Constraint (8) represents sub-tour elimination constraints, which ensure that a job cannot be the immediate predecessor or successor of two or more different jobs at the same time.

$$\sum_{i \in J_k^{'}} \sum_{j \in J_k^{'}} x_{ijk} \le | J_k^{'} | - 1 \qquad J_k^{'} \subseteq J_k \tag{8}$$

Note that any solution with a zero ARPI value is optimal. The relationship of PMWBSDS with what is in the literature is as follows:

- In Yildirim et al. (2007) and Duman et al. (2008), the objective function is minimization of the total completion time on each machine, i.e.,

  $$\min C_{total} = \sum_{k \in K} C_k .$$

  Furthermore, balancing the workload is achieved by having the following constraints:

  $$C_k \le \frac{1}{|K|} C_{total}(1+\alpha) \text{ and } C_k \ge \frac{1}{|K|} C_{total}(1-\alpha) \text{ for } k \in K$$

  where α is the level of tolerance above and below the average workload. This problem is very similar to vehicle-routing problems with load-balancing constraints. When there is one machine, the problem reduces to $1|S_{ij}|C_{max}$, which is NP-complete.

- When $S_{ijk}=0$, i.e., there is no sequence-dependent setups, and PMWBSDS reduces to the problem defined by Rajakumar et al. (2004, 2006).

- When there are machine-dependent setups, i.e., $S_{ik} = S_{ijk}$ and there is a constraint that enforces all machines to have the same completion time, i.e., C_k is constant then the problem reduces to the one proposed by Aubry et al. (2008). Aubry et al. prove that this problem is NP-complete by reducing the 3-partioning problem into this problem.

The PMWBSDS problem is closely related to the set partitioning problem: PMWBSDS aims to partition jobs into subsets (i.e., assign jobs to machines) and then sequence the jobs in such a way that all machines have the same completion time. However, since there are sequence-dependent setups between jobs, one must determine the "correct completion time" while having a non-delay schedule in order to achieve "zero" relative imbalance, if possible. PMWBSDS has an exponential number of possible solutions. This motivated us to develop heuristics, an ant colony optimization algorithm and a genetic algorithm to find good solutions in a reasonable amount of time.

3. ANT COLONY OPTIMIZATION FOR IDENTICAL PARALLEL-MACHINE SCHEDULING

This study proposes an ant colony optimization for parallel-machine scheduling with load balancing and setups algorithm (or simply ACO algorithm) to solve the PMWBSDS problem defined in section 2. The ACO algorithm is a metaheuristic that can be used to solve complex optimization problems (Dorigo and Gambardella, 1996). The ACO algorithm is developed by the inspiration of ants' ability to find the optimal (shortest) route between their nests and target food locations. In the presence of alternative routes, ants initially spread out to each route equally; however, after a certain period of time, they converge to the optimal route since there is more pheromone on shorter routes when compared to longer routes as a result of a higher number of trips occurring on shorter routes in a unit of time.

In standard ACO, the ants' route preferences depend on the amount of pheromone, which is determined by the usage frequency of the matching route. Shorter distance to the destination (i.e., better objective function value) results in greater pheromone level. The paths that correspond to potential solutions have pheromone level as a function of the route performances. Pheromone update has two main elements: evaporation of the pheromone on all of the routes at a certain rate, and keeping the amount of pheromone on the routes that the ants have already passed (i.e., the past solutions) inversely proportional to the total relative imbalance level. The evaporation ratio causes a reduction in the importance of the previous solutions. A pheromone increase that is inversely proportional to the fitness of the route ensures the importance of the fine solutions (Dorigo and Gambardella, 1997).

In load balancing in parallel machines with the sequence-dependent setups problem, the goal is to assign and sequence $|J|$ jobs over $|K|$ machines to minimize the total imbalance. To solve PMWBSDS using a metaheuristic, two decisions should be made: first, determining the assignment of jobs to machines, and then determining the order of jobs in order to obtain a good solution that will minimize the relative imbalance.

Construction of Graph

The graph that is used in ACO is generated as follows: jobs are represented as supernodes (which can also be defined as node clusters). Each supernode has $|K|$ nodes, which represent the machines on which each job can be processed. The nodes on each supernode are not connected. However, every node in a super node is connected to all other nodes in other supernodes. For the sake of completeness, a dummy node, which can be viewed as the ant's nest (i.e., the start and end point of an ant's tour), is defined. This dummy node is connected to every other node on the graph. As a result, the graph has one dummy node, $|J|$ supernodes, and $|J||K|$ nodes. The total number of edges are $(|J||K|)^2$. On this graph, to construct a solution, the artificial ant travels to each supernode (i.e., visits any node [machine] on that supernode) and then returns back to the dummy node (i.e., completes the tour on the graph). This tour spans $|J|+1$ edges. When an ant completes a tour, the order of visiting each supernode yields the order of assignment of jobs to the machines. Furthermore, the node visited at each supernode determines the assignment of a job to a machine.

************************Insert Figure 1 around here************************

An illustration for graph construction is given in Figure 1. In this example there are three jobs and two machines. In the graph, the jobs (supernodes) are represented by rectangles. The machines on which each job can be processed are represented by circles (nodes). The nodes in a rectangle are not connected. The dummy node (node 0) is connected to any other node on the graph. The goal of an ant is to visit all of the rectangles using the edges defined on the network and return back to the dummy node. This tour corresponds to a solution for PMWBSDS.

************************Insert Figure 2 around here************************

To illustrate this route construction, consider Figure 2, where there are five jobs and three machines. The path of Ant 1, represented by red dotted arcs, is (0), (1, 3), (3, 1), (2, 2), (4,1), (5, 3), (0), where (i, k) is (job, machine). This route results in the following solution: on machine 1, first job 3 and then job 4 are processed. On machine two, only job 2 is processed. Finally on machine three, first job 1 and then job 5 are processed. Similarly, artificial Ant 2's tour (represented by the solid line) results in the following solution: on machine 1, job 2 and then job 1 are processed. Machine 2 processes job 5, whereas machine 3 processes job 4 and then job 3.

Outline of Ant Colony Optimization Algorithm

In the ACO algorithm, after the construction of the graph, at each iteration, artificial ants travel over the graph to find routes that correspond to better solutions. Upon completion of an iteration, i.e., Υ ants complete tours of $|J|$ supernodes, the amount of pheromone trail on each edge is recalculated according to the quality of solutions obtained during that iteration.

A feasible order and assignment is obtained when an ant visits all of the supernodes. In order to complete a tour (i.e., find a solution to the PMWBSDS problem), at any given node, an ant must decide what node to visit next.

For any ant γ, the set of candidate nodes that may be visited in the remainder of the tour are defined as a tabu list. In $tabu^{\gamma}$, those jobs (supernodes) that have been visited (i.e., the jobs that have been assigned already) are excluded from the choice through the use of a tabu list.

Let $\tau_{((i,k),(i',k'))}(t)$ be the artificial pheromone trail on an arc from (i,k) to (i',k') at iteration *t*. Similarly, $\Delta\tau_{((i,k),(i',k'))}(t)$ is the incremental pheromone amount, which is found as a function of the fitness of the solutions that are found by all ants at iteration *t*. Initially, the pheromone level on each edge is set to an arbitrary but small level of τ_0. After an iteration is completed, the level of pheromone level on each arc is updated.

Let $ARPI^{\gamma}(t)$ be the average relative percentage imbalance for the solution obtained by ant γ at iteration *t*. Then, after iteration *t* is completed, the pheromone level on ant γ's path is updated using the following formula:

$$\Delta\tau^{\gamma}_{((i,k),(i',k'))}(t)=\begin{cases}\dfrac{1}{ARPI^{\gamma}(t)} & \text{if ant } \gamma \text{ travels on edge } ((i,k),(i',k'))\\ 0 & \text{otherwise}\end{cases}$$

Note that since the best objective function value for the load-balancing problem is zero, when a solution with zero *ARPI* is obtained, then the ant colony algorithm stops. The overall change in the pheromone level as a result of the ants (new solutions) is calculated as

$$\Delta\tau_{((i,k),(i',k'))}(t)=\sum_{\gamma=1}^{\Upsilon}\Delta\tau^{\gamma}_{((i,k),(i',k'))}(t)$$

The total amount of pheromone level at any edge on the network is

$$\tau_{((i,k),(i',k'))}(t+1)=(1-p)\tau_{((i,k),(i',k'))}(t)+\Delta\tau_{((i,k),(i',k'))}(t)$$

where *p* is an evaporation rate that represents the evaporation of the trail between iteration *t* and $t+1$. Before the algorithm starts, the pheromone level on arcs that correspond to the heuristic solutions is updated.

Selection of the next node, which has not been visited yet, can be made in three ways:

(a) With probability *q,* select the node that has the maximum pheromone amount, i.e.,

$$(i',k')=\underset{(\bar{i},\bar{k})\in tabu^{\gamma}}{\arg\max}\left\{\left[\tau_{((i,k),(\bar{i},\bar{k}))}(t)\right]\right\}$$

(b) With probability *d,* select the node with respect to a discrete probability function, which is a function of pheromone trails on all edges, i.e.,

$$\Pr^{\gamma}_{((i,k),(i',k'))}(t)=\begin{cases}\dfrac{\left[\tau_{((i,k),(i',k'))}(t)\right]}{\sum\limits_{(\bar{i},\bar{k})\in tabu^{\gamma}}\left[\tau_{((i,k),(\bar{i},\bar{k}))}(t)\right]}, & \text{if } (i,k)\in tabu^{\gamma}\\ 0, & \text{otherwise}\end{cases}$$

where $\Pr^{\gamma}_{((i,k),(i',k'))}(t)$ is the transition probability from node (i,k) to node (i',k') for the γ-th ant.

(c) With probability *r,* select the next node randomly from the tabu list. The random selection helps in diversifying the solutions.

Note that $q+d+r=1$.

The pseudocode for the ACO algorithm is summarized in Figure 3. To improve the performance of the algorithm, after each iteration, a first improvement local search procedure can be utilized (ACO*): the 2-exchange heuristic is used to switch the positions of two jobs, either on the same machine or on different machines. If the exchange improves the current solution, it is kept. At each iteration, $|J|^2$ exchanges is performed.

***********************Insert Figure 3 around here************************

4. A GENETIC ALGORITHM TO SOLVE THE PMWBSDS PROBLEM

Genetic algorithms mimic the process of evolution in order to solve complex combinatorial problems, and have been applied widely to parallel-machine problems. In each generation, a genetic algorithm is capable of producing and maintaining a set of feasible solutions (Fowler et al., 2003), maintaining a population of candidate solutions, and evaluating the quality of each candidate solution according to the problem-specific fitness function. New candidate solutions are created by selecting relatively fit members and recombining them through various genetic operators (crossover, mutation and selection). The pseudo-code of the genetic algorithm is presented in Figure 4, and then the components of the proposed genetic algorithm are explained in detail.

***********************Insert Figure 4 around here************************

Representation (Coding): Our implementation includes a super chromosome of size |*J*|, which is composed of sub-chromosomes that represent the sequence of jobs on each machine (see Figure 5 for a problem involving ten jobs on three machines). First jobs from the first machine are listed in the scheduled order on the super chromosome, followed by jobs on machine 2 through machine |*K*|. In another array, is kept the information on how many jobs are scheduled on each machine (e.g., for the solution in Figure 5, the array will be [3,4,3], i.e. three jobs on machine 1, four jobs on machine 2 and three jobs on machine 3).

**********************Insert Figure 5 around here************************

Initialization: Fowler et al. (2003) reported that assigning jobs based on some pre-determined rules may provide better solutions and reduce computational time than generating random solutions. Hence, an initial set of solutions includes solutions from the proposed heuristics and random solutions.

Evaluation of Fitness Function: The fitness function f for a chromosome Ω to penalize the objective function severely can be calculated as a function of the objective function value as

$$f(\mathrm{ARPI}_{\Omega}) = \mathrm{ARPI}_{\Omega}$$

where ARPI_{Ω} is the average relative percentage imbalance of the solution/chromosome Ω.

Selection: The selection model should reflect nature's survival of the fittest. Normally, in a genetic algorithm, chromosomes with a better fitness value will receive more chances to survive in the next generations. In this paper, the roulette wheel system is used to select parents for the next generation (Ip et al., 2000).

Crossover: Once parents are selected, the crossover operation is applied with a probability of *Pc* to generate two new offspring solutions. The type of crossover employed in our genetic algorithm implementation is single point crossover: Two parent strings are selected randomly from the population. A random number between 2 to $|J|$-1 is generated (where $|J|$ is the length of the chromosome) to determine the crossover point. When crossover is finished, the genes before the crossover point in the first chromosome are the first part of the first child chromosome. The second part of the first child chromosome is generated by checking the genes from the second chromosome one by one and adding those genes that are not yet in the child chromosome (Figure 6). Similarly, the 2nd child chromosome is generated.

***********************Insert Figure 6 around here************************

Mutation: The mutation operator moves a gene to another position on the chromosome randomly, with a probability equal to the mutation probability, *Pm*. If the new position is on the same machine, then only the order of jobs changes. However, if the gene is moved to another machine, the number of jobs and order of jobs on both machines may change. In Figure 7, job 5 on machine 1 is moved to the last job on machine 3, and the number of jobs is increased to three.

**********************Insert Figure 7 around here************************

Local Search (optional): After each iteration, similar to ACO search algorithm, a first improvement local search, a 2-exchange heuristic, is used to switch positions of the two jobs, either on the same machine or on different machines. If the exchange improves the current solution, it is kept. At each iteration, $|J|^2$ number of exchanges is performed.

Two variants of the genetic algorithm without (GA) and with local search (GA*) are utilized to solve the PMWBSDS problem.

5. OTHER METHODS TO SOLVE THE PMWBSDS PROBLEM

In order to provide quick solutions to the PMWBSDS problem, heuristics can be utilized. Although heuristics/dispatching rules cannot guarantee optimality, they are usually easier to implement, and they find solutions in less time compared to finding the optimal solution via methods such as branch and bound, or good solutions via metaheuristic methods. To solve the parallel-machine workload-balancing problems, several heuristics have been proposed. For example, Rajakumar et al. (2004, 2006) utilized Shortest Processing Time (SPT), Longest Processing Time (LPT) dispatching rules to schedule jobs on machines for the workload-balancing problem in the absence of setups. When there are sequence-dependent setups, Yildirim et al. (2007) proposed several dispatching rules that minimize the total completion time while considering the workload balancing constraint.

In this section, several heuristics similar to those of Yildirim et al. (2007) are presented to minimize the average relative imbalance in the presence of sequence-dependent setups. In these heuristics, jobs are ordered with respect to an ordering criterion, and then they are assigned to a machine based on an assignment criterion.

The criteria utilized to order jobs are as follows:

Random (RN): The order of jobs is random.

Longest Processing Time (LPT): Jobs are ordered in non-increasing processing time.

Shortest Processing Time (SPT): Jobs are ordered in non-decreasing processing time.

After the order is determined, then jobs selected from the ordered list are assigned to machines using the following assignment rules:

Setup Avoidance (SA): This rule assigns the job to the machine that causes the least setup time. Fowler et al. (2003) noted that the SA rule is a commonly used rule by scheduling practitioners for problems with sequence-dependent setups when the objective is to minimize the makespan.

Cumulative Processing Time (CPT): Jobs are assigned to the machine that yields the least cumulative workload.

Hybrid Cumulative Processing Time and Setup Avoidance (CPT-SA): At any iteration, if the imbalance is within α, then the SA rule is used. Otherwise, CPT is applied. In the case of a tie, jobs are assigned to the lower index machine. Note that, for any heuristic, the imbalance for machine *k* is calculated as

$$\alpha_k = 1 - \frac{C_k}{C_{\max}}$$

There are three ordering rules and three assignment rules. A combination of an "order" rule and an "assignment" rule results in nine heuristics to minimize the imbalance. As can be seen in Table 1, each heuristic, h_i, has a name, which consists of an ordering rule and an assignment rule. For example, with h_4, the LPT-SA heuristic orders the jobs via the longest processing time rule and then assigns jobs to the machines using the setup avoidance rule.

***********************Insert Table 1 around here************************

6. COMPUTATIONAL EXPERIMENTATION

This section discusses how to generate different scenarios to test the effectiveness of the proposed heuristics in environments of two to six machines. The ACO algorithm, GA, and heuristics are programmed using MATLAB R2007b. Below is a description of the experimental design, which is followed by results and discussion.

6.1 EXPERIMENTAL DESIGN

Generation of Experimental Data

The experimental setup considers the following factors: number of machines—two to six machines (2M, 3M, 4M, 5M, and 6M); number of jobs (20, 40, and 60); and processing time/setup time ratio (ρ : low [ρ =0.1], medium [ρ =1], and high [ρ =10]). As a result, the total number of scenarios is 45. For each scenario, we generated 20 problems and ran each instance ten different times. Nine solutions from heuristics h1,…, h9 are also considered in the set of initial solutions for both ant colony optimization and genetic algorithms.

The job data, processing time, and setup time are generated using a method similar to that of Fowler et al. (2003). As shown in Table 2, when the processing time/setup time ratio (*pt/st*) is high (i.e., *pt/st=0.1),* then the processing times are generated using a uniform distribution with a mean of 10 (U[0, 20]+5), while the setup time is generated using a uniform distribution that has an expected value of 150. The setup time matrix is asymmetric (i.e., s_{ijk} may not be equal to s_{jik}).

***********************Insert Table 2 around here************************

An initial experimentation is performed to determine the best imbalance factor that should be utilized for heuristics h3, h6, and h9, where the assignment rule is CPT-SA. The maximum imbalance factor is chosen to be 20%.

Fine Tuning of Parameters for ACO Algorithm

Selection of good parameters can play a vital role in the running time and effectiveness of the ACO algorithm. In our ACO algorithm implementation, the ant population size is 20, and the initial pheromone level is 10^{-8}. In order to determine the best combinations for *p, q, d,* and *r,* and also to investigate the impact of these parameters on ACO algorithm performance, an experimental design is performed. The ACO algorithm is run using *p, q, d,* and *r* at the levels summarized in Table 3.

***********************Insert Table 3 around here************************

The experimental design summarized in Table 3 is used to solve "difficult" problems with a *pt/st* ratio of 10, while keeping the number of machines constant. For each number of jobs (20, 40 and 60) and machine combination (2, 3, 4, 5, and 6), the experimentation is performed over 15 randomly generated problems using 42 combinations of *(p, q, d, r)* parameters. In all experiments, the ACO algorithm is run for 1,000 iterations and each experiment is run 100 times. The performance of a particular combination is determined by the average performance over 15 problems and 100 runs. The optimal parameter set for a problem class is the parameter combination that yields the best overall performance (see Table 4). For example, on a 40-job problem with two machines, the ACO algorithm finds good results with all combinations of (*p, q, d, r*). The best combination for 40-job problems on five machines is by having an evaporation rate of 0.5 while selecting the edge with the maximum pheromone level 80% of the time.

***********************Insert Table 4 around here************************

Fine Tuning of Parameters for GA Algorithm

The crossover probability (*Pc)* and mutation probability *(Pm)* of the GA proposed in section 4 is obtained using a similar experimental design utilized for the ACO algorithm parameter tuning. The initial population is from the nine heuristics and also randomly generated. GA has a population size of 20, and total number of iterations of 2,000.

The GA algorithm is run using *Pc* of 0.5, 0.7, and 0.9 and *Pm* of 0.005, 0.010, and 0.050. The experimentation is performed over the same data set, number of iterations and runs as in the ACO parameter fine tuning. The optimal combination of these parameters is summarized in Table 5 for different number of jobs and machines combinations. For example, for 40 jobs 5 machines, the optimal crossover probability is 0.9 and mutation probability is 0.01.

************************Insert Table 5 around here*************************

Note that we have performed fine tuning process for ACO and GA only on 20, 40 and 60 jobs.

6.2 RESULTS AND DISCUSSION

Heuristics h2, h3, h5, h8, and h9 also provide relatively good results for the 20-machine case. In these heuristics, the utilized assignment rules are CPT and CPT-SA. Among the heuristics, the best performance is obtained in h5, where the ordering rule is LPT. The heuristics (h1, h4, and h7) that utilize the SA assignment rule, which provides good results in minimization of completion time in the presence of sequence setups (Fowler et al., 2003), do not outperform other heuristics when the objective is the minimization of total relative imbalance. The ACO* algorithm finds results with a much lower ARPI compared to all heuristics and ACO, GA, GA* metaheuristics. However, one should note that the running time of the heuristics is negligible when compared with the running time of the metaheuristics. ACO* determines these results in less than two minutes of CPU time on a Pentium Dual Core Machine with 4 GB of memory and 120 GB of hard drive using

MATLAB R2007b as the programming medium. The CPU time increases with the number of jobs and the number of machines. When there are two machines, ACO* and GA* determine the optimal result (i.e., a solution with zero imbalance) almost instantaneously in most of the runs.

********************Insert Table 6 *********************

In all experimental settings, the dispatching rules (i.e., h1,..., h9) provide results worse than the ACO and GA since both metaheuristics utilize the solutions from dispatching rules as initial solutions. In the 20-job case (Table 6), the performance of metaheuristics is significantly improved when a local search (2-exchange) is added to improve the solutions. The ACO* performs the best with an average ARPI value of 0.427, while GA* improves the solution of GA from 1.280 to 1.103.

********************Insert Table 7 *********************

When the number of jobs is 40 (Table 7), we have similar observations as in the case of 20 jobs. However, when pt/st=10, in the four- and six-machine cases, GA* outperforms ACO*. The performance of the metaheuristics in increasing order is ACO* (0.381), GA* (0.464), ACO (1.004), and GA (1.198).

********************Insert Table 8 *********************

When the number of jobs is 60 (Table 8), the GA* outperforms ACO* only in the five-machine case, where processing times are relatively shorter than setup times (i.e., pt/st=0.1). Generally, the ARPI values for metaheuristics are GA≤ACO≤GA*≤ACO*.

When the number of jobs increases to 40 or 60, observations made for the 20-job case still hold. However, one should note that as the number of jobs increases, on average the solution quality for heuristics gets closer to the results provided for ACO. However, we still observe that heuristics h2, h5, h8, and h9 outperform the other heuristics. As a result, the CPT assignment rule is the best assignment rule when compared with SA or CPT-SA.

Table 9 summarizes the results for the ACO algorithm under different environments. Generally, when the number of machines increases, the ARPI value increases. When the number of jobs increases, usually the ARPI decreases. The performance of the ACO algorithm does not change when the processing time/setup time ratio changes.

***********************Insert Table 9 around here************************

When all problems are analyzed, ACO* performs better when the number of jobs increases. In addition, the quality of the solution improves as well (i.e., gets closer to the lower bound of zero). However, when the number of machines increases, the problems become more difficult. In other words, the ACO algorithm produces results with higher relative imbalance values. When the *pt/st* value increases, we cannot observe a direct relation in the quality of the relative imbalance (Table 9). When the relative performance of each heuristic with respect to ACO*, the best known solution, is analyzed, it is observed that when the number of machines increases, the relative performance of heuristics improves. This may be due to the inability of ACO* to obtain "perfect solutions" (i.e., solutions that have zero imbalance) as the number of machines increases.

Figure 8 presents box plots for ARPI values of different heuristics and metaheuristic algorithms. When the number of jobs increases, the ARPI value decreases significantly for h2, h5, and h8. The variability in ARPI increases when the number of jobs increases for h3 and h4. The metaheuristics, particularly ACO* algorithm, are quite robust as a function of number of jobs. Their performance improve when the processing time/setup time ratio is high. In general, when the number of machines increases, the ARPI increases for all heuristics. Furthermore, when the number of machines increases, variability in ARPI values increases for h2, h5, h8, and the metaheuristics. Note that, in general, h1, h4, and h7, in which the assignment rule is SA, have the most variability in results and also perform the worst, while h3, h6, and h9 have mediocre performance. Heuristics h2, h5, and h8, which utilize the CPT rule as the assignment rule, are the best among all heuristics. The ACO* algorithm, is better than all heuristics. Note that the SA assignment rule is widely used in practice (Fowler et al., 2003) for parallel-machine scheduling with setups for the minimum-completion-time objective. However, it is observed that this rule cannot be used to solve problems with load balancing in parallel-machine scheduling.

***********************Insert Figure 8 around here***********************

***********************Insert Table 10 around here**********************

When total completion time of the optimal ARPI solutions is analyzed, the dispatching rules and ACO* generally provide better results than other metaheuristics, since all metaheuristics were designed to find solutions for the ARPI problem not for the total completion time problem. In finding the best solution with ACO* and other metaheuristics, we selected the solution with the best total completion time among all alternative solutions available at that time (i.e., solutions having the same ARPI value).

The quality of the solutions obtained by the heuristics and the ACO algorithm when the objective is minimization of total completion time is presented in Table 10. When the output is analyzed, the general trend is as follows: as the number of jobs increases, the total completion time increases; and when the number of machines increases, the total completion time usually decreases. Results show that the quality of the solutions provided by the ACO algorithm has the highest total completion time, but ACO* has improved the total completion time very significantly. The heuristics that utilize the setup avoidance rule as the assignment rule provide better results compared to other heuristics. It can also be observed that the minimization of total relative imbalance and the minimization of total completion time are not equivalent load balancing policies.

7. CONCLUSIONS

In this study, a mixed-integer mathematical programming model for scheduling of parallel machines with sequence-dependent setups was proposed, where the objective is to minimize the average relative imbalance on all machines. Due to the computational complexity involved in solving the mathematical model, heuristics and metaheuristic algorithms were developed to generate solutions within a reasonable period of time. Heuristics were also used to generate the initial set of solutions for the ant colony optimization algorithm and genetic algorithm. Extensive computational experimentation was performed to test the effectiveness of the developed methodology over different scenarios. It was observed that the ant colony optimization algorithm with local search improves the heuristic and metaheuristics solutions significantly. Furthermore, the most promising heuristics are those that utilize the CPT rule as the assignment rule. It was observed that the initial order of the assignments does not have a significant impact on the total relative imbalance. In addition, computational experimentation showed that the results obtained by the ant colony optimization algorithm with local search for the PMWBSDS are inferior solutions for the load-balancing problem defined in Yildirim et al. (2007), where the objective is the minimization of the total completion time in the presence of load-balancing constraints.

A direct extension of this research would be to have a multi-objective mathematical model in which both minimization of total completion time and minimization of imbalance are two objectives to be considered. A multi-objective genetic algorithm approach could be developed to solve this problem as part of future research. Another extension would be to determine a lower bound to test the effectiveness of the ACO algorithm. A more detailed experiment to determine if there is a clear pattern between the parameters of the experimental design and the objective could be designed.

References

1. Allahverdi, A., Gupta, J. N. D., and Aldowaisan, T. A review of scheduling research involving setup consideration. *International Journal of Management Science*. 1999. 27, pp. 219-239.
2. Aubry, A., Rossi, A., Espinouse, M.L., and Jacomino, M. Minimizing setup costs for parallel multi-purpose machines under load-balancing constraint. *European Journal of Operations Research*. 2008. 187, pp. 1115-1125.
3. Chen J. F. Minimization of maximum tardiness on unrelated parallel machines with process restrictions and setups. *International Journal of Advanced Manufacturing Technology*. 2006. 29, pp. 557-563.
4. Dietrich, B. L., and Escudero, L. F. On solving a 0-1 model for workload allocation on parallel unrelated machines with setups. *Proceedings of Third ORSA/TIMS Conference on Flexible Manufacturing Systems: Operations Research Models and Applications*. 1989. pp. 181-186.
5. Dorigo, M., and Gambardella, L.M. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*. 1997. 1(1), pp. 53–66.
6. Dorigo, M., Maniezzo, V., and Colorni, A. Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*. 1996. 26(1), pp. 29–41.
7. Duman, E., Yildirim, M. B., and Alkaya, A. F. Scheduling continuous aluminum casting lines. *International Journal of Production Research*. 2008. 46 (20), pp. 5701-5718
8. Fowler, J. W., Horng, S. M., and Cochran, J. K. A hybridized genetic algorithm to solve parallel machine scheduling problems with sequence dependent setups. *International Journal of Industrial Engineering*. 2003. 10(3), pp. 232-243.
9. Gascon, A. and Leachman, R. C. A dynamic programming solution to the dynamic, multi-item, single-machine scheduling problem. *Operations Research*. 1998. 36 (1), pp. 50-56.
10. Hillier, M. S., and Brandeau, M. L. Cost minimization and workload balancing in printed circuit board assembly. *IIE Transactions*. 2001. 33 (7), pp. 547-557.

11. Ip, W.H., Li, Y., Man, K.F., Tang, K.S.. Multi-product planning and scheduling using genetic algorithm approach. Computers & Industrial Engineering. 2000. Vol. 38(2), pp. 283-296.
12. Pinedo, M. *Scheduling: Theory, Algorithms and Systems*. Springer Series in Operations Research and Financial Engineering, 1995.
13. Rajakumar, S., Arunachalam, V. P., and Selladurai, V. Workflow balancing strategies in parallel machine scheduling. *International Journal of Advanced Manufacturing Technology*. 2004. 23, pp. 366-374.
14. Rajakumar, S., Arunachalam, V. P., and Selladurai, V. Workflow balancing in parallel machine scheduling with precedence constraints using genetic algorithm. *Journal of Manufacturing Technology Management*. 2006. 17(2), pp. 239-254.
15. Rajakumar, S., Arunachalam, V. P., and Selladurai, V. Workflow balancing in parallel machine scheduling using genetic algorithm. *International Journal of Advanced Manufacturing Technology*. 2007. 33(11-12), pp. 1212-1221.
16. Reeves, C.R. Modern heuristic techniques for combinatorial problems. 1995. McGraw-Hill Book Company Inc., Europe.
17. Tahar, D. N., Yalaoui, F., Chu, C., and Amodeo, L. A linear programming approach for identical parallel machine scheduling with job splitting and sequence-dependent setup times. *International Journal of Production Economics*. 2006. 99 (1-2), pp. 63-73.
18. Tamaki H., Hasegawa Y., Kozasa J., and Araki M. Application of search methods to scheduling problem in plastics forming plant: A binary representation approach. *Proceedings of the 32nd IEEE Conference on Decision and Control*. 1993. pp. 3845-3850.
19. Yalaoui, F., and Chu, C. An efficient heuristic approach for parallel machine scheduling with job splitting and sequence-dependent setup times. *IIE Transactions*. 2003. 35(2), pp. 183-190.
20. Yamamoto, K. and Naito, S. A study on schema preservation by crossover. Systems and Computers in Japan. 2002. 33(2), pp. 64-76.
21. Yildirim, M. B., Duman, E., Krishna, K., and Senniappan, K. Parallel machine scheduling with load balancing and sequence dependent setups. *International Journal of Operations Research*. 2007. 4(1), pp. 1-8.

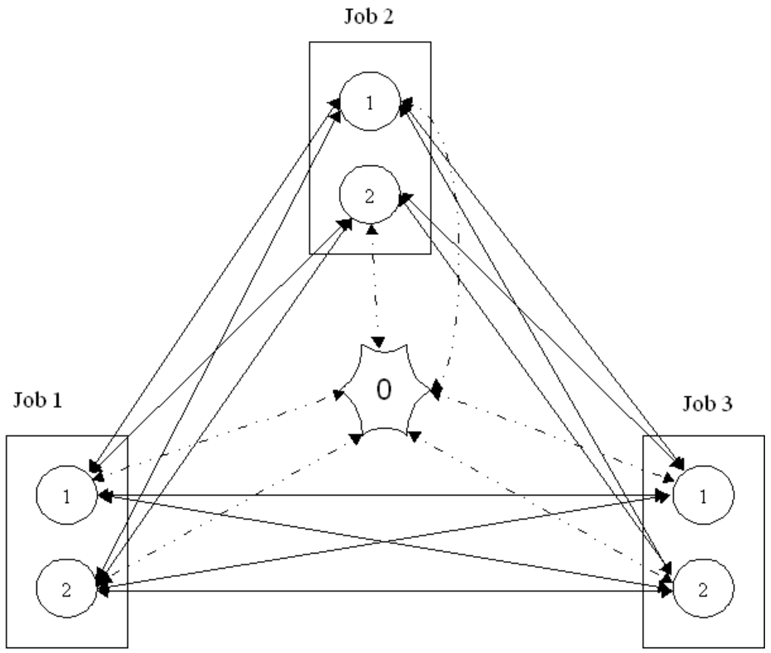

Figure 1. Network representation of load-balancing problem on ACO.

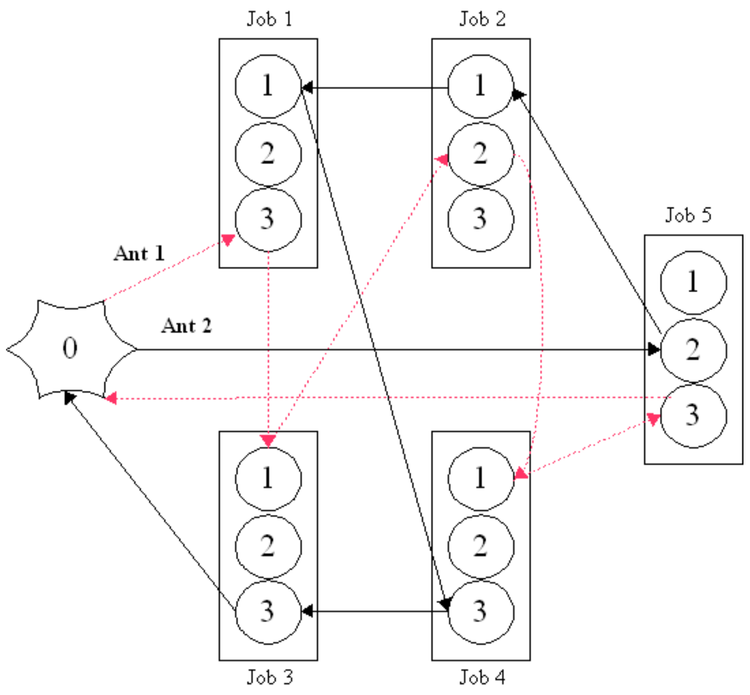

Figure 2. Tours for Ant 1 and Ant 2 in ACO.

```
Ant Colony Optimization Algorithm
STEP 0:  Initialize:
         a) Set iteration counter $t$=0.
         b) Set pheromone level on all edges to $\tau_0$.
STEP 1:  Construct solutions for each ant $\gamma$ (i.e., for $\gamma = 1$ to $\Upsilon$):
         Generate a tour by adding an edge from $tabu^{\gamma}$ using the following:
         a) With probability $q$, the node with the maximum pheromone level.
         b) With probability $d$, a discrete probability distribution with edge
            probability $\Pr^{\gamma}_{((i,k),(i',k'))}(t)$.
         c) With probability $r$, random selection.
STEP 2:  Local search (Optional)
STEP 3:  Evaluate the fitness of each solution.
STEP 4:  Keep the best solutions in a list. If not optimal (i.e., ARPI is not zero) and $t$
         < MaxIterationCount, then update the pheromone levels on the network via
         evaporation and incremental pheromone update processes.
STEP 5:  Set $t$=$t$+1.  Go to Step 1.
```

Figure 3. Pseudocode for ACO algorithm.

```
Genetic Algorithm
STEP 0:   Generate an initial population
STEP 1:   Evaluate the fitness value of the chromosomes
STEP 2:   Perform selection operation and give those individual that have better
          fitness values a more chance to survive in the next generation.
STEP 3:   Perform crossover and mutation operations according to probabilities.
STEP 4:   Perform local search (optional)
STEP 5:   Repeat steps 1, 2, 3 and 4 until the GA is run for the predetermined number
          of generations or an optimal solution with zero ARPI value is found
STEP 5:   Select the best chromosome.
```

Figure 4. Pseudocode for the genetic algorithm.

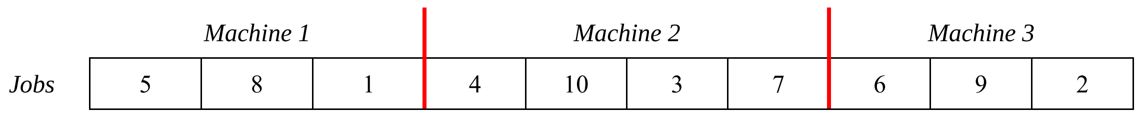

Figure 5. Representation of a chromosome

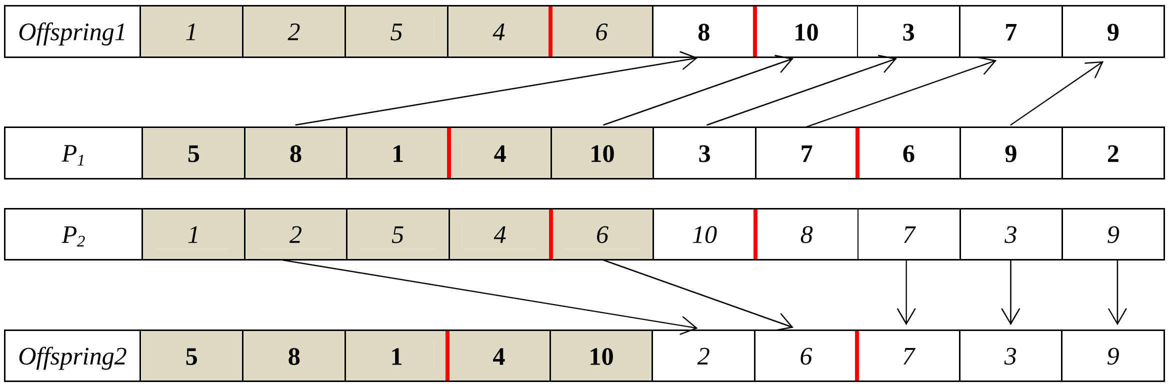

Figure 6. Crossover operation

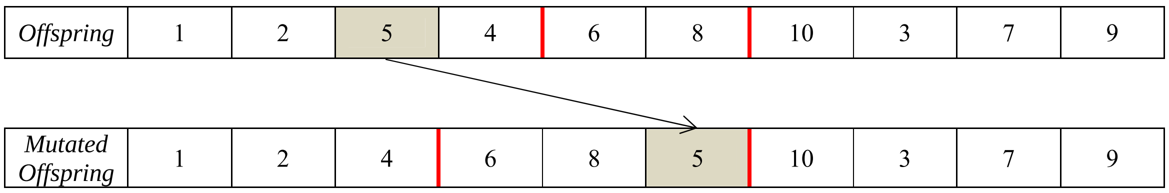

Figure 7. Mutation operator

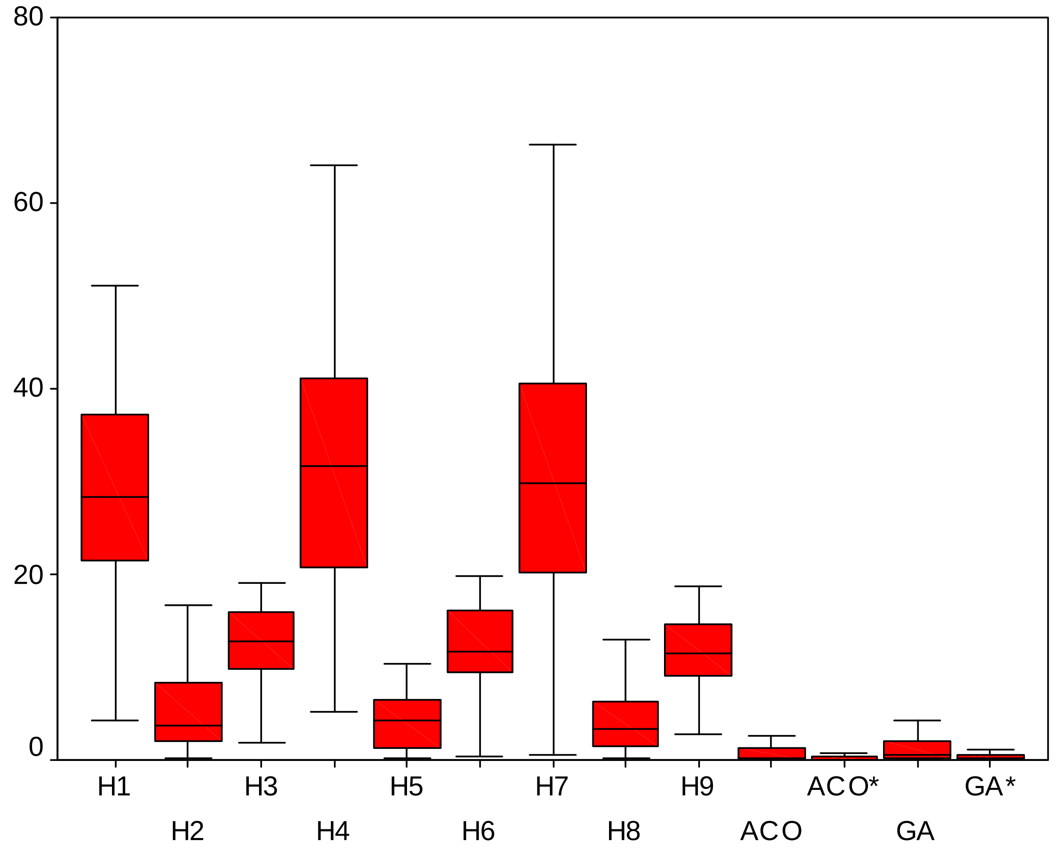

Figure 8. Average relative percentage imbalance values for different heuristics.

Table 1: Heuristics to minimize load balance in parallel machines with sequence-dependent setups

	Assignment Rule		
Ordering Rule	**SA**	**CPT**	**CPT- SA**
RN	h_1 RN-SA	h_2 RN-CPT	h_3 RN- CPT- SA
LPT	h_4 LPT-SA	h_5 LPT-CPT	h_6 LPT- CPT- SA
SPT	h_7 SPT-SA	h_8 SPT-CPT	h_9 SPT- CPT- SA

Table 2: Parameters for generation of processing and setup times

pt/st	**Ratio**	**Processing Time**	**Setup Time**
High	0.1=15/150	U[0, 20]+5	U[0, 7]*43
Moderate	1.0= 50/50	U[0, 20]+40	U[0, 7]*15
Low	10 = 70/7	U[0, 20]+60	U[0, 7]*2

Table 3: Parameter optimization experimentation design for ACO algorithm

p	q (best)	d (probabilistic)	r (random) = (1-(q+d))
(0.1, 0.3, 0.5)	0.8	(0.1, 0.2)	(0.0, 0.1)
(0.1, 0.3, 0.5)	0.6	(0.1, 0.2, 0.3, 0.4)	(0.0, 0.1, 0.2, 0.3)
(0.1, 0.3, 0.5)	0.4	(0.1, 0.2, 0.3, 0.4)	(0.2, 0.3, 0.4,0.5)
(0.1, 0.3, 0.5)	0.2	(0.1, 0.2, 0.3, 0.4)	(0.4, 0.5, 0.6, 0.7)

Table 4: Optimal (*p, q, d, r*) parameter combination for ACO algorithm

Machines	Jobs		
	20	40	60
2M	(0.5, 0.8, 0.1, 0.1)	All Combinations Optimum	All Combinations Optimum
3M	(0.5, 0.8, 0.1, 0.1)	(0.5, 0.6, 0.1, 0.3)	(0.3, 0.8, 0.2, 0.0)
4M	(0.5, 0.8, 0.1, 0.1)	(0.3, 0.2, 0.4, 0.4)	(0.1, 0.2, 0.1, 0.7)
5M	(0.1, 0.8, 0.1, 0.1)	(0.5, 0.8, 0.1, 0.1)	(0.3, 0.8, 0.1, 0.1)
6M	(0.1, 0.8, 0.1, 0.1)	(0.3, 0.6, 0.2, 0.2)	(0.3, 0.4, 0.3, 0.3)

Table 5: Optimal (Pc, Pm) parameter combination for GA algorithm

Machines	Jobs		
	20	40	60
2M	(0.7, 0.01)	(0.9, 0.01)	(0.9, 0.01)
3M	(0.9, 0.01)	(0.9, 0.01)	(0.9, 0.01)
4M	(0.7, 0.01)	(0.9, 0.05)	(0.9, 0.01)
5M	(0.9, 0.05)	(0.9, 0.01)	(0.9, 0.05)
6M	(0.9, 0.05)	(0.9, 0.05)	(0.9, 0.05)

Table 6: ARPI values when |J|=20

pt/st	M	h1	h2	h3	h4	h5	h6	h7	h8	h9	ACO	ACO*	GA	GA*
0.1	2	11.495	3.063	9.562	5.169	1.182	14.130	20.124	4.315	15.023	0.094	0.008	0.122	0.033
	3	27.037	3.761	12.192	49.916	4.200	5.100	30.934	14.505	18.721	0.158	0.011	0.787	0.595
	4	51.156	9.513	14.538	55.901	9.245	16.356	47.129	6.212	9.608	0.762	0.139	2.095	1.114
	5	64.971	16.425	16.439	56.785	18.915	12.645	50.683	7.982	15.574	1.933	0.597	3.607	1.753
	6	70.863	18.356	18.341	64.014	26.340	32.287	59.425	24.997	13.614	4.069	0.824	6.400	2.567
1	2	43.815	5.322	16.616	17.090	0.781	9.924	20.483	2.641	10.327	0.017	0.006	0.087	0.025
	3	33.025	3.319	8.782	39.073	9.293	13.409	27.547	1.263	17.602	0.194	0.043	0.720	0.362
	4	41.263	16.286	16.675	31.933	15.845	11.750	40.511	11.986	8.662	0.623	0.157	3.001	0.626
	5	25.939	13.584	18.985	38.413	19.373	27.655	21.323	11.505	9.389	1.324	0.427	3.827	1.789
	6	39.092	16.681	16.711	31.248	15.502	16.695	41.210	14.591	16.934	2.571	0.616	4.088	1.560
10	2	6.245	0.501	10.888	24.287	0.467	16.977	21.989	0.227	14.730	0.053	0.031	0.064	0.061
	3	4.352	6.090	17.285	17.553	5.111	18.562	38.728	4.470	4.362	0.065	0.021	0.617	0.046
	4	32.061	4.057	17.086	38.505	1.529	14.801	49.011	1.891	2.837	0.213	0.064	0.448	0.091
	5	33.418	8.436	6.578	35.242	5.426	19.376	59.364	1.960	17.119	0.310	0.032	0.647	0.176
	6	45.134	15.445	14.091	54.044	17.526	19.848	66.210	12.953	16.152	6.884	3.466	7.873	5.771
Mean		35.325	9.389	14.318	37.278	10.049	16.634	39.645	8.100	12.710	1.285	0.429	2.292	1.105

Table 7: ARPI values when |J|=40

pt/st	M	h1	h2	h3	h4	h5	h6	h7	h8	h9	ACO	ACO*	GA	GA*
0.1	2	21.728	1.960	2.624	11.122	0.718	6.419	13.077	3.629	10.819	0.006	0.005	0.040	0.007
	3	17.628	2.383	12.549	14.878	2.361	8.530	31.798	4.498	3.006	0.128	0.025	0.471	0.053
	4	40.955	5.194	11.740	34.049	6.203	10.338	37.988	6.567	10.879	0.902	0.087	1.021	0.290
	5	33.556	6.097	11.008	44.260	10.368	13.977	29.731	11.487	9.231	1.952	0.404	2.608	0.520
	6	26.817	13.708	10.617	50.657	4.824	16.462	44.668	17.074	10.861	2.153	1.089	2.549	1.631
1	2	23.233	1.951	13.149	8.641	0.276	9.924	7.308	3.304	0.211	0.086	0.004	0.017	0.006
	3	24.523	4.829	6.496	21.458	1.837	10.428	33.490	3.676	8.499	0.106	0.023	0.173	0.069
	4	15.270	10.640	14.398	15.904	6.386	8.464	29.028	4.927	12.262	0.856	0.135	0.514	0.204
	5	36.448	8.206	15.917	24.939	9.121	8.427	9.743	2.530	7.297	0.868	0.124	0.996	0.440
	6	37.265	11.709	17.645	23.752	16.143	10.635	41.562	5.844	14.594	3.748	1.036	4.338	1.665
10	2	7.809	0.299	3.747	31.721	0.927	0.352	16.358	0.972	12.296	0.004	0.011	0.095	0.024
	3	16.056	3.145	15.924	16.554	3.629	17.231	19.784	0.276	14.790	0.084	0.034	0.178	0.039
	4	28.927	1.402	16.228	23.057	0.732	12.058	27.882	1.197	15.562	0.173	0.065	0.353	0.083
	5	38.891	3.696	10.773	32.598	2.907	10.004	17.863	0.965	15.574	0.322	0.075	0.823	0.110
	6	34.556	6.373	14.119	40.342	6.393	16.047	18.144	4.662	12.621	3.741	2.680	3.800	1.973
Mean		26.911	5.439	11.796	26.262	4.855	10.620	25.228	4.774	10.567	1.009	0.387	1.198	0.474

Table 8: ARPI values when |J|=60

pt/st	M	h1	h2	h3	h4	h5	h6	h7	h8	h9	ACO	ACO*	GA	GA*
0.1	2	22.466	0.595	14.545	26.500	0.141	13.379	29.935	1.672	7.631	0.042	0.007	0.040	0.009
	3	21.545	2.406	14.552	39.478	3.186	7.993	40.359	1.712	13.592	0.097	0.009	0.220	0.034
	4	26.794	4.540	12.825	41.497	2.804	7.470	29.940	1.875	9.167	0.323	0.098	0.389	0.132
	5	28.358	3.112	12.775	43.979	4.933	9.595	28.706	6.366	13.495	1.718	0.659	1.019	0.343
	6	31.914	9.259	12.878	41.159	5.692	13.008	49.726	7.076	11.502	3.519	0.454	3.283	0.483
1	2	14.595	0.109	2.788	22.428	1.236	2.617	1.715	0.378	9.968	0.022	0.008	0.022	0.009
	3	23.637	1.761	2.831	33.118	2.892	6.025	12.870	2.590	5.903	0.101	0.021	0.116	0.059
	4	24.800	1.789	6.994	15.867	5.119	10.650	44.746	1.443	11.537	0.638	0.084	0.260	0.157
	5	38.764	1.976	14.831	42.295	6.461	16.691	38.660	2.754	15.414	1.061	0.390	1.242	0.563
	6	46.006	4.717	12.028	51.472	5.787	9.966	34.218	4.964	8.391	1.562	0.347	1.603	0.455
10	2	5.308	0.529	1.926	20.816	1.318	14.409	0.635	1.380	0.673	0.027	0.007	0.019	0.013
	3	17.963	2.644	2.991	18.209	0.853	10.854	28.907	0.666	11.350	0.064	0.005	0.182	0.016
	4	21.414	2.629	9.862	23.205	1.842	11.923	12.708	1.346	14.544	0.456	0.011	0.578	0.022
	5	34.163	1.584	12.799	17.460	1.299	10.060	25.469	1.097	13.225	0.217	0.056	0.340	0.084
	6	30.414	3.091	11.406	31.363	2.206	9.349	37.532	2.635	14.734	1.036	0.276	1.164	0.453
Mean		25.876	2.716	9.735	31.256	3.051	10.266	27.742	2.530	10.742	0.725	0.162	0.698	0.189

Table 9: Performance of ACO algorithm under different conditions

pt/st	0.1					1					10				
job\M	2	3	4	5	6	2	3	4	5	6	2	3	4	5	6
20	0.008	0.011	0.139	0.597	0.824	0.006	0.043	0.157	0.427	0.616	0.031	0.021	0.064	0.032	3.466
40	0.005	0.025	0.087	0.404	1.089	0.004	0.023	0.135	0.124	1.036	0.011	0.034	0.065	0.075	2.680
60	0.007	0.009	0.098	0.659	0.454	0.008	0.021	0.084	0.390	0.347	0.007	0.005	0.011	0.056	0.276

Table 10: Total completion time for different heuristics

Jobs	M	h1	h2	h3	h4	h5	h6	h7	h8	h9	ACO	ACO*	GA	GA*
20	2	1498	1489	1528	1483	1560	1484	1455	1485	1469	1617	1444	1553	1564
	3	1428	1505	1487	1432	1566	1487	1428	1473	1501	1533	1456	1540	1518
	4	1449	1482	1470	1433	1545	1469	1419	1476	1473	1538	1449	1544	1529
	5	1435	1472	1463	1429	1568	1483	1406	1535	1472	1510	1471	1543	1527
	6	1410	1444	1475	1453	1536	1467	1404	1499	1486	1504	1527	1519	1524
40	2	2942	3069	2958	2950	2974	2935	2875	2996	2982	2983	2779	3019	3020
	3	2826	3055	2916	2943	3044	2861	2840	2940	2956	3034	2781	3031	3033
	4	2844	2965	2874	2836	2994	2865	2822	2971	2870	3069	2775	3042	3031
	5	2824	2938	2915	2776	2929	2831	2806	3031	2902	2995	2776	3021	3046
	6	2729	3019	2918	2771	3069	2942	2840	3034	2868	3009	2778	3002	3026
60	2	4358	4495	4417	4488	4564	4426	4304	4594	4430	4576	4207	4576	4580
	3	4386	4427	4372	4445	4684	4317	4350	4435	4422	4511	4205	4596	4580
	4	4354	4465	4483	4261	4470	4396	4296	4570	4410	4665	4202	4569	4595
	5	4282	4582	4396	4184	4397	4259	4280	4590	4332	4509	4201	4587	4607
	6	4259	4636	4340	4322	4591	4396	4377	4439	4284	4516	4196	4558	4600