PARALLEL MACHINE SCHEDULING TO MINIMIZE THE MAKESPAN WITH SEQUENCE DEPENDENT DETERIORATING EFFECTS

Alex J. Ruiz-Torres*

Facultad de Administración de Empresas
Universidad de Puerto Rico – Río Piedras
San Juan, PR 00931-3332, USA
E-mail: alex.ruiztorres@upr.edu
Phone: (787) 764-0000 x87149

Giuseppe Paletta

Dipartimento di Economia e Statistica
Universita` della Calabria (UNICAL),
87036 Rende, CS, Italy
E-mail: g.paletta@unical.it
Phone: +39-0984-492428

Eduardo Pérez

Ingram School of Engineering
Texas State University
601 University Drive San Marcos, TX 78666
E-mail: eduardopr@txstate.edu
Phone: (512) 245-1826

ABSTRACT. A new unrelated parallel machine scheduling problem with deteriorating effect and the objective of makespan minimization is presented in this paper. The deterioration of each machine (and therefore of the job processing times) is a function of the sequence of jobs that have been processed by the machine and not (as considered in the literature) by the time at which each job is assigned to the machine or by the number of jobs already processed by the machine. It is showed that for a single machine the problem can be solved in polynomial time, whereas the problem is NP-hard when the number of machines is greater or equal than two. For the last case, a set of list scheduling algorithms and simulated annealing meta-heuristics are designed and the effectiveness of these approaches is evaluated by solving a large number of benchmark instances.

Keywords. Multiprocessor scheduling, unrelated parallel machines, machine and job deterioration, simulated annealing meta-heuristic.

1. INTRODUCTION

Research that addresses the scheduling of deteriorating jobs has gained significant popularity in the last two decades. The tenant of problems with deteriorating jobs is that the processing time of the jobs is a function of their start time or the number of jobs since the start of the schedule (or since a maintenance activity), which is again related to the time since the start of the schedule. This paper addresses a variant of the job deterioration problem that considers the case where the deterioration of the processing time for a job depends on the specific jobs that have been previously processed by the machine. This perspective is in line with Yang [1] and Yang et al. [2], where the jobs are not per se deteriorating, but instead the machines are the ones deteriorating, although this differentiation is not made in most models. In our model the deterioration of the machines (and therefore of the job processing time) is a function of the sequence of jobs that have been processed by the same machine and not a function of the two approaches reported in the literature: the time at which the job is assigned to the machine or the number of jobs already processed. Our version of the problem is not yet addressed, and is highly relevant in many practical cases.

Two examples of the proposed relationship between deterioration and job assignment are presented next. The first is the assignment of construction jobs to "work gangs" during a shift. Each job has a baseline processing time, related to the time when all the workers are "fresh". As the workers perform each job they become increasingly tired and therefore its processing speed deteriorates, but this deterioration depends on the particular job sequence. Let us say there are four independent non-sequential jobs, each taking a baseline time of 2 hours (each would take 2 hours if done first thing in the morning): dig a trench in a hard ground, demolish a shed, clean a storage area, and paint a wall (ordered by effort). While performing the jobs from hardest to easiest may require 9.4 hours, performing the jobs from easiest to hardest may require 8.5 hours. In the first sequence, the workers may get tired from having performed the first two jobs and therefore take longer time in performing the easy ones. On the other hand, when performing the easy tasks first, they will be "fresh" to complete the exhausting ones.

The second example, similar to that described by Yang et al. [2], considers a shop where machines are used to process a material, for example cutting stock or shredding wood. It can be assumed that depending on the material hardness the tools deteriorate differently. If the jobs with the "softer" material are processed first, the tools will deteriorate less, therefore the tools will maintain a higher level of performance. On the other hand, if the "hard" material jobs are performed first, the tools will deteriorate "faster" and completing the tasks on the softer material jobs take longer, for example if the machine has to be run slower to assure it properly performs the shredding process.

The remaining of the paper is organized as follows. In the next section, we discuss the recent literature on deteriorating jobs in the parallel machine environment. In section 3, we formulate the problem for the unrelated machines, show some properties, and provide an illustrative example. In section 4, we present special problem cases, while in section 5 we present some heuristics, based on the proprieties showed in section 3, for the unrelated machines case. An experimental analysis is presented in Section 6. Section 7 concludes the paper and provides suggestions for future research.

2. LITERATURE REVIEW

There has been considerable interest in the problem of deteriorating jobs since the seminal work by Gupta and Gupta [3] and Browne and Yechiali [4]. Reviews of the literature for deteriorating job problems have been completed by Alidaee and Womer [5] and by Cheng et. al. [6]. In this section we focus on recent papers in the deteriorating job problem that consider parallel machine~~s~~ environment.

A research stream in the literature characterizes processing time as a function of the job's start time. Let us define p'_j, p_j, t_j, and b_j as the actual processing time, 'baseline processing time', deterioration factor, and start time of job j respectively. Kang and Ng [7] propose a fully polynomial approximation scheme for the parallel machine problem with makespan objective where the process time is modeled by $p'_j = p_j + b_jt_j$. Kuo and Yang [8] and Toksari and Güner [9] address a variant of the linear version where the increasing (or decreasing) rate is identical for all the jobs (thus $b_j = b$ for all jobs). Kuo and Yang [8] consider the sum of the completion time for all jobs and the sum of the machine completion times as measures of performance, and demonstrate for two linear functions that the problems are polynomially solvable. Toksari and Güner [9] address the objective of minimizing the earliness/tardiness with a common due date. They design a mathematical model for the problem and analyze its performance for solving large problems. Mazdeh et al. [10] consider the parallel machine problem with job deterioration of the form $p'_j = p_j + b_jt_j$ concurrently with the cost of machine deterioration based on the allocation of jobs to the different machines. The authors consider the joint minimization of the total tardiness and the machine deterioration cost. Given the problem is NP-Hard the authors propose a heuristic algorithm and test its effectiveness.

Several researchers address the parallel machine problem when $p'_j = p_jt_j$ with the objective of minimizing the makespan. Ren and Kang [11] present polynomial approximation algorithms for the problem and provide the complexity of the two machine case. Ji and Cheng [12] solve the sum of job completion times problem, while Ji and Cheng [13] address the makespan and sum of machine completion times criteria, proposing approximation algorithms. Cheng et al. [14] also address the makespan, but also consider the maximization of the minimum machine completion time. Given both problems are NP-hard, the authors propose heuristic algorithms and evaluate their performance. Huang and Wang [15] address two uncommon objectives: total absolute differences in completion times and the total absolute differences in waiting times. They demonstrate these problems are solvable by polynomial algorithms.

A second research stream in the literature characterizes the processing time as a function of the job's position in the machine sequence. Let us define p'_{jrh} as the processing time of job j if processed in the r^{th} position of machine h. The papers by Yang [1] and Yang et al. [2] consider the parallel machine problem where the processing time is defined by one of two models $p'_{jrh} = p_{jh} + r \times b_{jh}$ and $p'_{jrh} = p_{jh} \times r^{b_{jh}}$, where b_{jh} is the deterioration effect of job j on machine h, and the position r depends on the number of jobs after a maintenance event. Both papers address the minimization of the total machine load taking into consideration the joint decisions of maintenance frequency and timing, and the assignment and sequence of the jobs on the machines. The article by Yang [1] deals with the identical parallel machine case, therefore there is no difference in base processing time or deterioration effects between machines, while Yang et al. [2] deal with the unrelated machines (a more general case). In both papers the authors demonstrate that all versions addressed with a given job frequency can be solved in polynomial time.

Mosheiov [16] addresses the general problem where p'_{jrh} is defined as a non-decreasing function in r and the processing time could be unique to each machine, therefore possibly requiring a n^2m input matrix of processing times (where n is the number of jobs and m the number of machines). For this problem the author provides a polynomial time algorithm and describes several extensions. Toksari and Güner [17] combine position based learning with linear and non-linear deterioration with the objective of minimizing the earliness/tardiness with a common due date. They design a mathematical model for the problem and provide a lower bound procedure to address larger problems. On a related problem where the deterioration is neither time dependent nor position dependent, Hsu et al. [18] consider the problem of unrelated parallel machines with rate modifying activities to minimize the total completion time, where at most one rate modifying activity can occur per machine. They propose an algorithm that can solve the problem in $O(n^{m+3})$ if the rate modifying activities are less than 1 (and greater than 0) and in $O(n^{2m+2})$ if the rate modifying activities are larger than 1.

3. THE PROBLEM

The problem under consideration can be stated as follows. There are n independent jobs $N=\{1,\dots,j,\dots n\}$ to be processed on m parallel machines $M=\{1,\dots,k,\dots m\}$. All the jobs are non-preemptive and available for processing at time zero. Each machine can process only one job at a time and cannot stand idle until the last job assigned to it has been finished. There are g possible positions in each machine, $g = n$, and let G be the set of positions. Let p_{jk} be the baseline processing time of job j on machine k. Let d_{jk} be the deteriorating effect of job j on machine k and $0 \leq d_{jk} < 1$ for all $j \in N$ and $k \in M$. Therefore, as in Hsu et al. [18] we include a rate modifying activity, but in our problem each job has a different rate modifying activity.

Let X_k be the ordered set of jobs assigned to machine k, and $x[h, k]$ be the job assigned to position h of machine k. Let q_{kh} indicate the performance level of machine k for the job in position h and let q_{kh} be defined by $q_{kh} = (1 - d_{x[h-1,k]k}) \times q_{k(h-1)}$ for each machine $k \in M$ and each position h greater than 1. It is assumed the machines start with no deterioration, thus $q_{k1} = 1$ for all $k \in M$. The actual processing time of the job $x[h, k]$ on machine k is equal to $p'_{x[h,k]k} = p_{x[h,k]k} / q_{kh}$. The problem under consideration is the assignment of jobs to the machines and to sequence the jobs on the machines so that the maximum completion time of all the jobs is minimized.

Let C_k be the completion time of all the jobs assigned to machine k, therefore the sum of the actual processing times for the jobs assigned to the machine. The considered measure of performance is the maximum completion time $C_{max} = max_{\,k\in M} \{C_k\}$. The complexity of this problem is clearly NP-hard given the problem that assumes identical machines and no deterioration ($P||C_{max}$) is well known to be NP-Hard.

The mathematical formulation for this problem is presented next. The decision variable x_{jkh}, $j\in N$, $k\in M$, $h\in G$, is a binary variable that is equal to 1 if job j is assigned to machine k in position h, 0 otherwise.

$$\text{Minimize } z = C_{max} \qquad (1)$$

$$\sum_{j\in N} x_{jkh} \leq 1 \qquad \forall\, h\in G,\, k\in M \qquad (2)$$
$$\sum_{h\in G,\, k\in M} x_{jkh} = 1 \qquad \forall\, j\in N \qquad (3)$$
$$\sum_{j\in N,\ h\in G} p_{jk}/q_{kh}\times x_{jkh} \leq C_{max} \qquad \forall\, k \in M \qquad (4)$$
$$x_{jkh} \leq \sum_{l\in N} x_{lk(h-1)} \qquad \forall\, j\in N,\, k\in M,\, h\in G\backslash\{1\} \qquad (5)$$
$$q_{kh} = \sum_{j\in N} (1-d_{jk})\times q_{k(h-1)} \times x_{jk(h-1)} \qquad \forall\, h\in G\backslash\{1\},\ k\in M \qquad (6)$$
$$q_{k1} = 1 \qquad \forall\, k \in M \qquad (7)$$
$$x_{jkh} \in \{0, 1\} \qquad \forall\, j\in N,\, k\in M,\, h\in G \qquad (8)$$

In the model, Equation (1) is the objective function. Equation (2) states that to each position in each machine can be assigned at most one job, while Equation (3) states that each job must be assigned just once to one position in one machine. Equation (4) establishes the total load in each machine must be not greater than C_{max}, while Equation (5) guarantees continuous assignments. Equations (6-7) define the performance level of each machine for each job position.

The following lemma establishes that for a given set of jobs assigned to a machine, say k, the minimum completion time C_k is found by sequencing the jobs on the machine in non-increasing order of the ratio $r_{jk}= p_{jk}(1-d_{jk})/d_{jk}$. Therefore, the proposed problem when $m = 1$ is solvable in polynomial time.

Lemma 1. Let X_k be the set of jobs assigned to machine k and ordered so that
$p_{x[1,k]k}(1-d_{x[1,k]k})/d_{x[1,k]k} \geq p_{x[2,k]k}(1-d_{x[2,k]k})/d_{x[2,k]k} \geq p_{x[3,k]k}(1-d_{x[3,k]k})/d_{x[3,k]k} \geq \ldots \geq p_{x[n_k,k]k}(1-d_{x[n_k,k]k})/d_{x[n_k,k]k}$,
where n_k is the number of jobs in X_k, then the **completion time** C_k is optimal.

Proof. Let X'_k be as set X_k but with the jobs in positions 2 and 3 exchanged.

The completion time C_k for X_k is:

$$C_k = p_{x[1,k]k} + p_{x[2,k]k} /(1 - d_{x[1,k]k}) + p_{x[3,k]k} /[(1 - d_{x[1,k]k})(1 - d_{x[2,k]k})] + \ldots + p_{x[y,k]k} /\prod_{h=1,n_k-1} (1-d_{x[h,k]k}).$$

The completion time C'_k for X'_k is:

$$C'_k = p_{x[1,k]k} + p_{x[3,k]k} /(1 - d_{x[1,k]k}) + p_{x[2,k]k} /[(1-d_{x[1,k]k})(1-d_{x[3,k]k})] + \ldots + p_{x[y,k]k}/\prod_{h=1,n_k-1} (1-d_{x[h,k]k}).$$

It is supposed, by contradiction, that $C_k > C'_k$, then:

$$p_{x[2,k]k} /(1-d_{x[1,k]k})+p_{x[3,k]k} /[(1-d_{x[1,k]k})(1-d_{x[2,k]k})] > p_{x[3,k]k} /(1-d_{x[1,k]k})+p_{x[2,k]k} /[(1-d_{x[1,k]k})(1-d_{x[3,k]k})].$$

It follows that

$$p_{x[2,k]k} + p_{x[3,k]k} /(1 - d_{x[2,k]k}) > p_{x[3,k]k} + p_{x[2,k]k} /(1 - d_{x[3,k]k}),$$

then

$$p_{x[2,k]k} - p_{x[2,k]k} /(1 - d_{x[3,k]k}) > p_{x[3,k]k} - p_{x[3,k]k} /(1 - d_{x[2,k]k}),$$

$$p_{x[2,k]k}[1 - 1/(1 - d_{x[3,k]k})] > p_{x[3,k]k} [1 - 1/(1 - d_{x[2,k]k})],$$

$$p_{x[2,k]k} [- d_{x[3,k]k} /(1 - d_{x[3,k]k})] > p_{x[3,k]k} [- d_{x[2,k]k} /(1 - d_{x[2,k]k})],$$

$$p_{x[2,k]k} [d_{x[3,k]k} /(1 - d_{x[3,k]k})] < p_{x[3,k]k} [d_{x[2,k]k} /(1 - d_{x[2,k]k})].$$

Finally

$$p_{x[2,k]k} [(1 - d_{x[2,k]k})/ d_{x[2,k]k}] < p_{x[3,k]k} [(1 - d_{x[3,k]k})/ d_{x[3,k]k}],$$

which cannot be true given

$$p_{x[2,k]k} [(1 - d_{x[2,k]k} / d_{x[2,k]k}]\geq p_{x[3,k]k} [(1 - d_{x[3,k]k})/ d_{x[3,k]k}].$$

It must be concluded that $C_k \leq C'_k$.

Q.E.D.

An example is used to illustrate the problem. We consider the case with m = 2 and n = 8. The job characteristics are presented in Table 1. Let schedule S1 be an arbitrary schedule with X_1 = {1-2-3-4} and X_2 = {5-6-7-8} (jobs are sequenced by job number), while schedule S2 maintains the same job assignment to machines, but the jobs are now sorted in non-increasing order of r_{jk}. Finally, schedule S3 is an optimal C_{max} schedule generated through full enumeration. All three schedules are presented in Figure 1, which illustrates the job information (job number and actual processing time), the completion time for each job as well as the machine performance level at the start of each job. By comparing schedules S1 and S2, the job order has an effect on the actual processing times of the jobs and the deterioration levels, thus on the completion times for the machines. It is obvious that the machine deteriorating levels at the end of the positions improve when comparing S1 and S2. The job assignments to the machines changed from S1 to S3, even when the actual processing times increased for specific jobs, for example job 2. The maximum completion time for schedules S1, S2, and S3 are 47.5, 46.7 and 39.2 respectively.

<Insert Table 1 and Figure 1 about here >

Special Case

There is an additional special case for the described problem when $m > 1$. As presented in Lemma 2, an optimal C_{max} solution can be found for the simple case with identical parallel machines, and identical processing times and deteriorations for all jobs.

***Lemma* 2.** We assume $p_{jk} = p$ and $d_{jk} = d$, $\forall\ j \in N$ and $\forall\ k \in M$. Let $f = \lceil n/m \rceil$ be the smallest integer not less than n/m. Let X_k be the set of jobs assigned to machine $k \in M$. An optimal makespan is found if each set X_k has at most f jobs.

Proof. If each set X_k has exactly f jobs (thus $f = n/m$) then the workload on each machine is $\lambda = p + p/(1-d) + p/[(1-d)^2] + \ldots + p/[(1-d)^{f-1}]$ and $C_{max} = \lambda$. It is obvious that if a job is moved across machines, one machine will have $f + 1$ jobs, and therefore another machine has $f - 1$ jobs. The machine with $f + 1$ jobs has a load of $\lambda + p/[(1-d)^f]$ and the machine with $f - 1$ jobs has a load of $\lambda - p/[(1-d)^{f-1}]$. Clearly, C_{max} increased in the new case by $p/[(1-d)^f]$, given $0 \leq d < 1$. By using the same approach when $f > n/m$, it is easy to demonstrate that the assignment of f jobs to $(m + n - f \times m)$ machines and of $(f - 1)$ jobs in to the remaining $(f \times m - n)$ machines is optimal for C_{max}.

4. SOLUTION APPROACHES

Given the problem complexity and relevance, we examine the performance of several heuristics to generate solutions (schedules) for the described problem. In this section we first present methods used to generate initial solutions followed by the implementation of a meta-heuristic.

4.1 List Schedules

The generation of ordered job lists is a common first step in scheduling algorithms. Eight approaches are used to generate job ordered lists. The lists are based on ordering by non-increasing value the following job characteristics:

- $p_j^{min} = min_{k \in M}\,\{p_{jk}\}$
- $p_j^{max} = max_{k \in M}\,\{p_{jk}\}$
- $d_j^{min} = min_{k \in M}\,\{d_{jk}\}$
- $d_j^{max} = max_{k \in M}\,\{d_{jk}\}$
- $r_j^{min} = min_{k \in M}\,\{p_{jk}(1-d_{jk})/d_{jk}\}$
- $r_j^{max} = max_{k \in M}\,\{p_{jk}(1-d_{jk})/d_{jk}\}$
- $v_j^{min} = min_{k \in M}\,\{p_{jk}/(1-d_{jk})\}$
- $v_j^{max} = max_{k \in M}\,\{p_{jk}/(1-d_{jk})\}$.

The generation of the schedule is based on assigning jobs from the ordered list to the machine with the smallest new load. When determining the machine loads, the optimal machine sequence is determined by sequencing jobs in the machine by non-increasing order of r_{jk}. The step by step procedure is presented next, where L is an ordered list of all jobs (by one of the described job characteristic).

Step 1. Remove first job from L, this is job j. Let $v = \infty$ and $k = 0$.
Step 2. Let $k = k + 1$.
Step 3 Add j to machine k. Sort the jobs in k by non-increasing order of r_{jk}.
Step 4 If $C_k < v$, then $f = k$ and $v = C_k$.
Step 5 Remove j from machine k. Sort the jobs in k by non-increasing order of r_{jk}.
Step 6. If $k < m$, then return to Step 2.
Step 7. Add j to machine f. Sort the jobs in f by non-increasing order of r_{jf}.
Step 8. If $L \neq \emptyset$, return to Step 1, else End.

The best solution obtained by using the previous procedure for each of the 8 ordered lists is called the seed schedule.

4.2 Simulated Annealing Meta-heuristic

Previous research has demonstrated the ability of probabilistic search techniques to find efficient/close to optimal solutions to complex scheduling problems [19]. One such technique is simulated annealing which has been used to tackle parallel machine problems in multiple cases [20]. The structure of SA algorithms is well known and based on the acceptance of "bad" solutions with a certain probability, with the objective of escaping local optima. SA is an iterative methodology, and as the number of iterations increases, the probability of accepting "bad" solutions decreases. The process stops when a limit to the number of cycles with no improvement is reached. The probability of accepting "bad" solutions is controlled by a process temperature. In the implementation of the SA meta-heuristic two search strategies are used. In the first case, called SA_1, the improvement is based solely in the change in makespan while in the second case, SA_2, we consider the overall change in workload, assuming this could have an effect on future exchanges.

Given the structure of SA algorithms is generally well known, we describe the most important details of the implementation.

Notation

S	the current schedule;
S^{best}	the current best schedule;
n_S	the number of jobs in the makespan machine of the current schedule S;
$Q_{initial}$	the initial temperature;

Q the temperature;
W the cooling parameter;
C_{max}^{V} the makespan of schedule V;
M_{sum}^{V} the sum of machine completion times of schedule V;
Δ^1_B the makespan improvement level for neighbor schedule B from current schedule S: $C_{max}^{S} – C_{max}^{B}$;
Δ^2_B the total machine completion time improvement level for neighbor schedule B from current schedule S: $M_{sum}^{S} – M_{sum}^{B}$;
ϕ_B number drawn from a (0,1) uniform distribution for neighbor schedule B;
y the loop counter.

Outline

Step 1. *Inputs*: Let S = Initial schedule, $S^{best} = S$.

Step 2. *Initialize parameters*: $y = 0$ and $Q = Q_{initial}$.

Step 3. *Neighborhood generation*

For each loop two types of neighbors are generated: pairwise exchanges between each job in the makespan machine and the jobs in all other machines, and single job reassignment where each job in the makespan machine is assigned to another machine. Given the completion time of each single machine k is optimally found by sorting the jobs by non-increasing order of r_{jk}, the search is limited to job to machine assignment and does not consider the particular positions in the machines. The total number of neighbors for a schedule is $n_S\,(n + m – 1 – n_S)$ given there will be $n_S\,(m – 1)$ single job insertions and $n_S\,(n – n_S)$ pairwise exchanges.

Step 4. *Acceptance of downhill move*

For each neighbor B the two improvement levels are determined (Δ^1_B and Δ^2_B). If $\Delta^1_B > 0$, this is a candidate neighbor.

SA_1. The candidate neighbor B with the highest value of Δ^1_B is selected, say B^*.

SA_2. The candidate neighbor B with the highest value of Δ^2_B is selected, say B^*.

If such a schedule exists and $C^{best}_{max} > C^{B^*}_{max}$ then $S^{best} = B^*$.

If such a schedule exists, then set $S = B^*$ and go to Step 2.

Step 5. *Acceptance of an uphill move*

For each neighbor B a random number drawn from a uniform interval distribution (0, 1) is determined, ϕ_B. If $\phi_B < exp\,(– \Delta^1_z / Q)$, this is a candidate neighbor. Select from the candidate neighbors the schedule with minimum ϕ_B, say B^*. If such a schedule exists, let $S = B^*$.

Step 6. *Loop control*: If $y < 2n$, then $y = y + 1$, $Q = Q \times W$, and go to Step 3. Else End.

Based on pilot experiments, we selected several control parameters for the implementation of the SA algorithms, realizing that computing time and resulting performance are sometimes at a tradeoff. A maximum of $2n$ temperature changes are considered, the cooling parameter (W) is set at 0.9 and the initial temperature control parameter $Q_{initial}$ is set at 4. All of these values are in line with previous applications of SA for similar scheduling problems [21]. The neighborhood size is $O(nm)$ for reassignments and $O(n^2)$ for interchanges. In case that no downhill move is found, the algorithm will accept an uphill move, which is the "trademark" of SA algorithms. In our implementation, the move that has the lowest random value and meets the "gate" based on how large is the deterioration is selected from all the neighbors. The algorithm stops once the loop control variable y reaches a value equal to $2n$.

Finally, three implementations of the SA algorithm are considered, the two previously mentioned; SA_1, SA_2, and the third case called SA*. The first two implementations use as the initial schedule S the seed schedule (i.e. the schedule with the lowest makespan among the schedules generated by the eight list scheduling rules presented in section 4.1). The third implementation, SA*, represents the use of the SA heuristic a total of 16 times, where each schedule in Section 4.1 is used as an initial schedule and two searches (SA_1 and SA_2) are performed. The best schedule generated from these sixteen searchers is the result of SA*. It is obvious here that SA* will require a significantly larger amount of computational time than either SA_1 or SA_2.

5. COMPUTATIONAL EXPERIMENTS

This section evaluates the performance of the described heuristics to generate optimal or efficient solutions to the problem. As in Gupta and Ruiz-Torres [22] we use two experimental sets: *OB* and *BB*. *OB* uses the optimal solution as the benchmark point, whereas *BB* uses the best solution found by the set of heuristics as benchmark since no optimal solution is available because of the size of the instances.

5.1 Experimental Parameters

We consider four experimental parameters: the number of machines (m), the number of jobs (n), the range of processing times (p_{gen}), and the range of the deteriorating effects (d_{gen}). The processing time p_{jk} is generated by a uniform distribution with range p_{gen}=(u_{min}, u_{max}), and the deteriorating effect d_{jk} by a uniform distribution with range d_{gen}=(d_{min}, d_{max}). For experiment set *OB* the evaluated levels of n are 8, 11, and 14, while the evaluated levels of m are 2, 3 and 4. For experiment set *BB* the evaluated levels of n are

20, 35, and 50, while the levels of m are 4, 7, and 10. For both *OB* and *BB* experiments, we consider two levels of the processing time range: (1, 100) and (100, 200), and two levels of the deterioration effect range: (1%, 5%) and (5%, 10%). Table 2 presents a summary of the two experiments. Twenty five replications are evaluated by experimental parameters, this gives a total of 1800 instances (900 for *OB* and 900 for *BB*). Instances are available at http://ruiz-torres.uprrp.edu/dm/.

<Insert Table 2 about here >

5.2 Results for *OB* experiment

We first analyze the performance of the schedules generated using the ordered lists described in Section 4.1 on the *OB* experiment. Table 3 presents the percentage of times each of the eight rules generates the seed schedule (lowest makespan schedule).

The percentages for each row adds to more than a 100% as often more than one of the ordered lists methods generate the best list schedule. The two rules with the highest percentages are $L(p_j^{min})$ and $L(v_j^{min})$, each generating close to 27%, while the two rules based on the deterioration parameter, $L(d_j^{min})$ and $L(d_j^{max})$, generate the lowest percentage at 13.9% and 15.2%, respectively. The results show that the experimental factors have an effect on which rule generates the best schedule. For example, at p_{gen} = (1,100) the schedules generated by $L(v_j^{min})$ represent 35.3% of the best schedules, while at p_{gen} = (100, 200), it represents 18.9%. This effect is consistent for all the rules where a smaller percentage of the best schedules is generated by each rule at p_{gen} = (100, 200). At p_{gen} = (1, 100) the average sum of all percentages is 215% (thus on average at least two rules generate the best schedule for an instance), while at p_{gen} = (100, 200), the average sum of percentages is 128%. While the d_{gen} and m parameters have no effect on the best schedule generation percentage, the number of jobs also has an effect on this combined measure. As n increases the sum of percentages decreases; at n = 8, the sum is 223%, while at n = 14 the average sum is 130.7%. For example, at p_{gen} = (1,100), d_{gen} = (1%, 5%), m = 4, and n = 8, each of the eight rules generates 36% or more of the best schedules with a sum of 328%, thus each of them is generated on average by three of the rules. On the other hand, at p_{gen} = (100, 200), d_{gen} = (5%, 10%), m = 3, and n = 14 the sum of percentages is 104%, thus in 24 out of 25 instances the best schedule is generated by only one of the rules. We conclude that the generation of schedules using the eight rules is warranted given the small computational effort required and the knowledge gained that under some conditions all rules have the potential to generate the best seed schedule.

<Insert Table 3 about here >

We next analyze the performance of the heuristics in terms of the error versus the optimal, and the percentage of times each of the heuristics find the optimal solution. The average results in terms of relative error $(C_{max}^{heuristic} - C_{max}^{Optimal}) / C_{max}^{Optimal}$ are presented in Table 4, while the percentage of times a heuristic found the optimal solution is presented in Table 5. To simplify the presentation only $L(p_j^{min})$,

$L(d_j^{min})$, $L(r_j^{min})$, and $L(v_j^{min})$ are presented; given these outperform the other four list scheduling versions for the overall experiment set. Next, the performance of the best solution (seed) found by list scheduling heuristics and the performance of SA_1, SA_2 and SA* are presented.

<Insert Table 4 and 5 about here >

The overall average error for the list scheduling heuristics (Table 4) range from 11.2% to 16.1%, while the seed schedule has an average error of 3.99%. The improvement obtained by the SA heuristics over the seed schedules is highly significant as the error for SA_1 and SA_2 is 0.88% and 0.86%, respectively. The error of SA* is very small at 0.012%, given this method found an optimal schedule in 892 out of 900 instances (Table 5). Furthermore, as can be observed in Table 5, the list scheduling heuristics perform poorly, since the best, $L(v_j)$, finds only 8.4% of the optimal solutions. The seed schedule is optimal in 22.6% of the instances, while the SA heuristics generate the optimal solution in 73.6% and 75.1% of the cases respectively for SA_1 and SA_2. Finally, as mentioned earlier, the SA* multi-search finds the optimal schedule for most of the instances, although clearly at a higher cost in computational time.

Table 6 presents the percentage of optimal results, summarized by experimental parameter, for the seed and SA heuristics. The p_{gen} parameter has an interesting effect, as the seed schedule performance decreases significantly as p_{gen} goes from (1,100) to (100, 200), the performance deteriorates slightly for SA_1 and SA_2, and does not change for SA*. The results for d_{gen} indicate a difference in performance between SA_1 and SA_2: at d_{gen} = (1%, 5%), SA_2 finds 2.4% more optimal solutions, and at d_{gen} = (5%, 10%) their performance is almost identical (difference < 1%). As m increases, the performance of the SA methods deteriorates, and deteriorates/improves for the seed schedule. As n increases the performance of all the methods decreases. The effect of m and n on SA_1 and SA_2 is similar; at the lowest level (m = 2, n = 8), SA_1 and SA_2 generate more than 80% of the optimal solutions, while at the highest levels (m = 4, n = 14) these generate less than 70% on average. The SA_1 and SA_2 heuristics perform similarly for most levels of m, except for m = 4 where SA_2 generates 2.3% more optimal solutions. In the case of parameter n, when n = 8, SA_2 finds 3% more optimal solutions than SA_1, while when n = 14, SA_1 finds 1% more optimal solutions (a reverse of "dominance"). This points out to a slight difference in performance for the two "fast" SA rules. Finally, the change in performance for SA* is very slight, with m being the parameter that presents the highest level of change in performance, at m =2 the heuristic SA* found 100% of the optimal solutions, while at m = 4, it finds 97.7% of the optimal solutions.

<Insert Table 6 about here >

The conclusion we draw from these results is that the single search SA approaches (SA_1 and SA_2) work well, finding close to optimal solutions for the range of experimental parameters analyzed, and that the multiple search SA approach (SA*) works extremely well, finding 99% of the optimal solutions. We note that finding the optimal solution through full enumeration required about two hours of CPU time for problems with m = 4 and n = 14, while heuristics required *insignificant* amounts of computing time. The list scheduling heuristics require fractions of a second for any of the instances, while the SA_1, SA_2 and SA* heuristics require for any of the instances a few seconds. The most time consuming heuristics is SA*, which required 25 seconds on average for the problems with m = 4 and n = 14, thus the time savings (versus 2 hours for a full search) are considerable. The experiments also demonstrate that performance of SA_1 and SA_2 is influenced by the experimental parameters, although the effect is minimal in regards to the performance of SA*.

5.3 Results for *BB* experiment

Table 7 presents the performance of the schedules generated using the ordered lists on *BB* experiment. As for the previous experiments, $L(p_j^{min})$ and $L(v_j^{min})$ generate the majority of the seed schedules, again generating close to 27%, while the two rules based on the deterioration parameter generate the lowest percentage at 7.3% and 8.8% respectively for $L(d_j^{min})$ and $L(d_j^{max})$. The results in terms of the sum of the percentages are consistent with those obtained for the *OB* experiment, where at p_{gen} = (1, 100) the sum of percentages is 147%, therefore each seed is generated by “1.5” of the rules. However, at p_{gen} = (100, 200) the sum of averages is 101%; in 15 out of 18 experimental points the sum is 100%, indicating that each seed is found by only one of the ordered lists. This reinforces the conclusion that generating schedules using the eight ordered list methods is valuable as in combination they generate a “good” initial solution with a minimal computational time requirement.

<Insert Table 7 about here >

SA* always generates the best solution (as the solutions generated by SA_1 and SA_2 are part of the solution set generate by SA*). The results presented in Table 8 are the error of the seed, SA_1 and SA_2 heuristics versus the best solution found (by SA*) and the number of times each of the three methods generates a schedule equal to that generated by SA*. The average error is 11.7%, 3.33%, and 3.7% respectively for the seed, SA_1, and SA_2 heuristics. As expected, the SA heuristics improve on the seed, but there is no significant difference between them. When we observe the percentage of “best” solutions found, the seed is equal to the best in only 3.56% of the cases, while in 16.89% and 14.56% of the cases the schedules obtained by SA_1 and SA_2 equal the best solution found. These results indicate that for *larger* problems, SA_1 and SA_2 will in general find a much smaller percentage of the optimal solutions than for *small* problems (as in experiment *OB*).

<Insert Table 8 about here >

The results for the error and percentage of best solutions found are summarized by experimental parameter in Table 9. When we consider the relationship between the experimental parameters and the average error, the error for the SA_1 and SA_2 heuristics generally worsens as m and n increase (problem size increases), and as p_{gen} changes from (100, 200) to (1, 100). Regarding the best solutions found, the number of jobs has the most significant effect: at n = 20, the SA_1 and SA_2 heuristics find about 30% of the best schedules, while at n = 50 the average is less than 10%, thus this clearly demonstrates again that the performance of the single search SA heuristics will be poor for problems with large n values. Two interesting results are that at m = 4, the SA_1 heuristic generates a higher percentage of best solutions (20% versus 13.7%), while at m = 10, SA_2 generates a larger percentage of the best solutions (16.3% versus 15%).

<Insert Table 9 about here >

Table 10 summarizes the resulting CPU times for the relative benchmark experiments by experimental parameter (all experiments were performed on a personal computer running on Intel Core Duo processor at 2.2GHz and 4GB RAM). The average CPU times for SA_1, SA_2, and SA* are 4.54, 4.46, and 441.1 seconds, respectively. The highest average CPU time requirement for an experimental point is at m = 4 and n = 50, where the SA_1, SA_2, and SA* heuristics require an average of 21.1, 20.75, and 2,019 seconds, respectively (not shown in the Table). The ratio of 100 to 1 in CPU time between the single search SA heuristics and SA* is observed through all the experimental combinations and can be noted in Table 10. While the p_{gen} and d_{gen} parameters have no effect on CPU times, as m increases CPU time decreases, while as n increases, CPU time increases. It is interesting to observe that a change in the value of parameter m from 4 to 10, a 250% change, reduces CPU time by a factor of 5, while a change in n from 20 to 50, also a 250% change, increases CPU time by 44 times. We conclude that implementing SA* for problems with large values of n would become unfeasible in terms of computational time requirements.

<Insert Table 10 about here >

6. SUMMARY AND FUTURE WORK

This paper proposes a new unrelated parallel machine scheduling problem that considers the minimization of the makespan when the deteriorating effect depends on the sequence of the jobs in the machines, and has designed a set of list scheduling algorithms and simulated annealing meta-heuristics. An extensive computational investigation serves to evaluate the performance of the proposed algorithms against optimal solutions and the best solutions found by the set of heuristics. The results of this study show that the multi-start simulated annealing meta-heuristic is capable of producing high quality solutions for a wide range of instances. Future research directions include the same problem with the minimization of other objective functions, for example the minimization of the total completion time of the jobs or the minimization of the earliness/tardiness as in Toskari and Guner [17].

ACKNOWELDGMENTS
This research was supported by a grant from the Facultad de Administración de Empresas of the University of Puerto Rico.

REFERENCES

[1] Yang, S.J. (2011) Parallel machines scheduling with simultaneous considerations of position-dependent deterioration effects and maintenance activities . *Journal of the Chinese Institute of Industrial Engineers* 28 (4), 270–280.

[2] Yang, D.L., Cheng, T.C.E., Yang, S.J., Shu, C.J. (2012). Unrelated parallel-machine scheduling with aging effects and multi-maintenance activities. *Computers & Operations Research* 39(7), 1458-1464.

[3] Gupta, J.N.D., Gupta, S.K. (1988). Single facility scheduling with nonlinear processing times. *Computers and Industrial Engineering* 14, 387–393.

[4] Browne, S., Yechiali, U. (1990). Scheduling deteriorating jobs on a single processor. *Operations Research* 38, 495–498.

[5] Alidaee, B., Womer, N. K. (1999). Scheduling with Time Dependent Processing Times: Review and Extensions. *Journal of Operational Research Society* 50, 711-720.

[6] Cheng, T.C.E., Ding, Q., Lin, B.M.T. (2004). A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research* 152, 1–13.

[7] Kang, L., Ng, C.T. (2007). A note on a fully polynomial-time approximation scheme for parallel-machine scheduling with deteriorating jobs. *International Journal of Production Economics* 109, 180-184.

[8] Kuo, W.H., Yang, D.L. (2008). Parallel-machine scheduling with time dependent processing times. *Theoretical Computer Science* 393, 204–210.

[9] Toksari, M.D., Guner, E. (2010). The common due-date early/tardy scheduling problem on a parallel machine under the effects of time-dependent learning and linear and nonlinear deterioration. *Expert Systems with Applications* 37, 92–112.

[10] Mazdeh, M.M., Zaerpour, F., Zareei, A., Hajinezhad, A. (2010). Parallel machines scheduling to minimize job tardiness and machine deteriorating cost with deteriorating jobs. *Applied Mathematical Modelling* 34, 1498-1510.

[11] Ren, C.R., Kang, L.Y. (2007). An approximation algorithm for parallel machine scheduling with simple linear deterioration. *Journal of Shanghai University* (English Edition) 11(4), 351-354.

[12] Ji, M., Cheng, T.C.E. (2008). Parallel-machine scheduling with simple linear deterioration to minimize total completion time. *European Journal of Operational Research* 188, 342-347.

[13] Ji, M., Cheng, T.C.E. (2009). Parallel-machine scheduling of simple linear deteriorating jobs. *Theoretical Computer Science* 410, 3761-3768.

[14] Cheng, T.C.E., Lai, P.J., Wu, C.C., Lee, W.C. (2009). Single-machine scheduling with sum-of-logarithm-processing-times-based learning considerations. *Information Sciences* 179, 3127-3135.

[15] Huang, X., Wang, M.Z. (2011). Parallel identical machines scheduling with deteriorating jobs and total absolute differences penalties. *Applied Mathematical Modelling* 35, 1349-1353.

[16] Mosheiov, G. (2012). A note: Multi-machine scheduling with general position-based deterioration to minimize total load. *International Journal Production Economics* 135, 523-525.

[17] Toksari, M.D., Güner, E. (2009). Parallel machine earliness/tardiness scheduling problem under the effects of position based learning and linear/nonlinear deterioration. *Computers & Operations Research* 36, 2394 -2417.

[18] Hsu, C.J., Cheng, T.C.E., Yang, D.L. (2011). Unrelated parallel-machine scheduling with rate-modifying activities to minimize the total completion time. *Information Sciences* 181, 4799-4803.

[19] Koulamas, C., Antony, S.R., Jaen, R. (1994). A survey of simulated annealing applications to operations-research problems. *OMEGA-International Journal of Management Science* 22, 41–56.

[20] Glover, F., Kochenberger, G.A. (2003). Handbook of Metaheuristics, Kluwer Academic Publishers, Dordrecht.

[21] Anderson, E.J., Glass, C.A., Potts, C.N. (1997). Machine scheduling. In: Aarts, E., Lenstra, J.K. (eds.) Local Search in Combinatorial Optimization, Wiley, Chichester.

[22] Gupta, J.N.D., Ruiz-Torres, A.J. (2001). LISTFIT heuristic for minimizing makespan on identical parallel machines. *Production Planning and Control* 12, 28-36.

Table 1. Job/resource information.

Job	p_{j1}	p_{j2}	d_{j1}	d_{j2}
1	15	11	0.05	0.03
2	9	12	0.06	0.02
3	8	6	0.04	0.05
4	12	16	0.04	0.08
5	6	9	0.07	0.04
6	10	13	0.06	0.09
7	12	7	0.07	0.04
8	13	8	0.04	0.02

Table 2. Summary of the experimental frameworks.

Experiment	n	M	p_{gen}	d_{gen}
OB	8, 11, 14	2, 3, 4	(1,100), (100,200)	(1%,5%), (5%,10%)
BB	20, 35,50	4, 7, 10	(1,100), (100,200)	(1%,5%), (5%,10%)

Table 3. Percentage of times a list schedule finds the seed on *OB* experiment.

p_{gen}	d_{gen}	m	n	$L(p_i^{min})$	$L(p_i^{max})$	$L(d_i^{min})$	$L(d_i^{max})$	$L(r_i^{min})$	$L(r_i^{max})$	$L(v_i^{min})$	$L(v_i^{max})$
(1, 100)	(1%, 5%)	2	8	32%	32%	28%	36%	36%	28%	36%	24%
			11	24%	32%	0%	12%	40%	20%	28%	40%
			14	20%	12%	24%	4%	28%	12%	24%	20%
		3	8	36%	40%	20%	28%	40%	24%	40%	44%
			11	28%	24%	24%	16%	32%	20%	28%	16%
			14	28%	20%	4%	4%	20%	20%	24%	28%
		4	8	40%	44%	40%	36%	36%	40%	40%	52%
			11	28%	24%	8%	20%	36%	12%	36%	28%
			14	32%	28%	16%	4%	32%	12%	28%	32%
	(5%, 10%)	2	8	56%	48%	8%	20%	52%	20%	56%	48%
			11	40%	32%	8%	20%	32%	32%	40%	32%
			14	28%	4%	8%	24%	20%	16%	28%	12%
		3	8	32%	28%	28%	24%	24%	36%	32%	28%
			11	24%	16%	12%	16%	24%	28%	24%	16%
			14	36%	8%	12%	0%	28%	20%	32%	4%
		4	8	72%	36%	24%	20%	48%	40%	68%	36%
			11	44%	24%	12%	16%	56%	20%	48%	16%
			14	24%	24%	0%	20%	28%	24%	24%	24%
(100, 200)	(1%, 5%)	2	8	20%	20%	16%	24%	8%	24%	20%	24%
			11	16%	20%	12%	16%	36%	4%	16%	0%
			14	8%	16%	12%	12%	16%	0%	20%	24%
		3	8	32%	20%	12%	24%	16%	16%	28%	20%
			11	32%	16%	12%	16%	8%	12%	24%	4%
			14	40%	8%	8%	4%	12%	8%	20%	12%
		4	8	16%	12%	8%	16%	28%	32%	20%	24%
			11	16%	20%	8%	8%	12%	24%	12%	20%
			14	8%	4%	16%	4%	20%	16%	20%	16%
	(5%, 10%)	2	8	32%	24%	24%	16%	8%	8%	36%	24%
			11	20%	4%	16%	24%	8%	16%	20%	4%
			14	16%	4%	20%	12%	4%	12%	16%	20%
		3	8	16%	28%	8%	12%	20%	24%	8%	28%
			11	16%	24%	20%	4%	20%	8%	8%	16%
			14	12%	0%	4%	4%	24%	16%	20%	24%
		4	8	24%	28%	4%	8%	12%	16%	28%	24%
			11	24%	12%	8%	8%	16%	16%	20%	16%
			14	4%	16%	16%	16%	8%	36%	4%	12%
Overall				**27.1%**	**20.9%**	**13.9%**	**15.2%**	**24.7%**	**19.8%**	**27.1%**	**22.6%**

Table 4. Average error for *OB* experiment.

p_{gen}	d_{gen}	m	n	$L(p_j)$	$L(d_j)$	$L(r_j)$	$L(v_j)$	seed	SA_1	SA_2	SA*
(1, 100)	(1%, 5%)	2	8	10.4%	12.1%	9.0%	9.8%	1.6%	0.1%	0.1%	0.00%
			11	10.3%	15.8%	9.5%	10.1%	1.8%	0.5%	0.5%	0.00%
			14	11.0%	12.1%	11.6%	10.9%	4.0%	0.5%	0.5%	0.00%
		3	8	10.8%	15.3%	10.4%	10.5%	3.3%	1.0%	0.8%	0.00%
			11	12.8%	17.9%	11.4%	12.8%	4.1%	1.0%	1.2%	0.00%
			14	14.2%	20.3%	17.7%	13.9%	6.9%	1.3%	1.1%	0.00%
		4	8	14.2%	15.4%	16.0%	14.2%	1.6%	1.1%	1.1%	0.02%
			11	13.9%	24.7%	15.7%	13.6%	6.4%	0.8%	0.7%	0.08%
			14	14.8%	26.6%	14.9%	14.5%	6.5%	3.5%	4.1%	0.04%
	(5%, 10%)	2	8	4.5%	16.7%	7.4%	4.5%	0.7%	0.0%	0.0%	0.00%
			11	9.5%	17.9%	7.7%	9.5%	1.5%	0.5%	0.3%	0.00%
			14	8.4%	12.7%	7.2%	7.9%	2.6%	0.2%	0.3%	0.00%
		3	8	12.1%	12.5%	15.9%	12.1%	0.8%	0.5%	0.3%	0.00%
			11	13.7%	25.3%	13.9%	13.2%	4.1%	1.4%	1.4%	0.00%
			14	11.5%	21.4%	11.1%	11.5%	4.0%	1.0%	1.4%	0.00%
		4	8	6.9%	21.4%	13.0%	6.9%	3.0%	1.1%	1.4%	0.00%
			11	11.3%	25.6%	11.0%	9.6%	3.6%	2.0%	1.6%	0.00%
			14	17.3%	28.0%	14.6%	17.3%	6.2%	2.4%	2.6%	0.01%
(100, 200)	(1%, 5%)	2	8	8.5%	9.8%	9.5%	7.5%	2.3%	0.5%	0.1%	0.00%
			11	8.7%	10.0%	8.1%	9.1%	3.6%	0.2%	0.2%	0.00%
			14	8.5%	9.5%	8.7%	8.1%	3.3%	0.2%	0.2%	0.00%
		3	8	9.2%	12.2%	9.9%	9.2%	3.4%	1.4%	1.0%	0.00%
			11	11.4%	12.2%	13.7%	10.5%	4.7%	0.7%	0.8%	0.07%
			14	10.5%	15.3%	15.4%	12.3%	6.7%	0.7%	0.7%	0.00%
		4	8	15.0%	20.1%	12.9%	15.7%	4.2%	0.8%	0.6%	0.00%
			11	12.0%	15.0%	12.9%	11.0%	3.6%	1.6%	1.3%	0.18%
			14	12.7%	14.0%	14.0%	12.1%	6.5%	1.0%	1.2%	0.00%
	(5%, 10%)	2	8	6.5%	8.9%	9.9%	6.8%	2.1%	0.0%	0.0%	0.00%
			11	7.6%	9.0%	8.0%	8.6%	3.4%	0.3%	0.3%	0.00%
			14	9.0%	10.0%	10.6%	8.1%	3.6%	0.0%	0.0%	0.00%
		3	8	12.2%	11.5%	12.0%	12.1%	3.7%	0.8%	0.5%	0.00%
			11	12.2%	14.6%	12.5%	11.9%	5.1%	0.8%	0.6%	0.00%
			14	13.3%	15.7%	12.1%	13.0%	6.4%	0.7%	0.9%	0.00%
		4	8	14.8%	17.4%	16.9%	13.5%	4.4%	0.8%	0.7%	0.00%
			11	15.1%	15.9%	12.8%	16.0%	7.1%	0.6%	0.7%	0.00%
			14	16.1%	15.6%	12.5%	14.8%	7.0%	1.6%	1.6%	0.03%
Overall				**11.41%**	**16.06%**	**11.96%**	**11.20%**	**3.99%**	**0.88%**	**0.86%**	**0.012%**

Table 5. Average number of optimal solutions found for *OB*

p_{gen}	d_{gen}	m	n	$L(p_j)$	$L(d_j)$	$L(r_j)$	$L(v_j)$	Seed	SA_1	SA_2	SA*
(1, 100)	(1%, 5%)	2	8	16%	20%	24%	20%	48%	96%	96%	100%
			11	4%	0%	12%	4%	32%	84%	88%	100%
			14	4%	4%	8%	4%	24%	88%	88%	100%
		3	8	24%	12%	20%	24%	48%	76%	80%	100%
			11	4%	16%	16%	4%	40%	72%	72%	100%
			14	8%	0%	0%	4%	16%	64%	76%	100%
		4	8	32%	32%	32%	32%	72%	80%	80%	96%
			11	16%	0%	8%	16%	28%	72%	76%	96%
			14	4%	4%	8%	4%	20%	40%	40%	96%
	(5%, 10%)	2	8	28%	0%	20%	28%	48%	96%	96%	100%
			11	8%	4%	8%	8%	32%	76%	84%	100%
			14	4%	4%	12%	8%	32%	92%	88%	100%
		3	8	20%	20%	16%	20%	64%	80%	84%	100%
			11	0%	4%	4%	0%	12%	64%	64%	100%
			14	12%	0%	4%	12%	28%	76%	64%	100%
		4	8	52%	24%	40%	48%	68%	84%	84%	100%
			11	24%	4%	24%	24%	52%	64%	72%	100%
			14	4%	0%	12%	4%	24%	64%	60%	92%
(100, 200)	(1%, 5%)	2	8	0%	4%	4%	0%	20%	92%	96%	100%
			11	0%	0%	0%	4%	4%	88%	88%	100%
			14	0%	0%	0%	4%	8%	84%	88%	100%
		3	8	4%	4%	4%	4%	20%	68%	72%	100%
			11	0%	0%	0%	0%	0%	68%	68%	96%
			14	0%	0%	0%	0%	0%	48%	48%	100%
		4	8	4%	8%	8%	4%	16%	76%	80%	100%
			11	0%	0%	0%	0%	8%	44%	52%	96%
			14	0%	0%	0%	0%	0%	52%	48%	100%
	(5%, 10%)	2	8	8%	12%	0%	16%	20%	100%	100%	100%
			11	0%	0%	0%	0%	0%	76%	76%	100%
			14	0%	0%	0%	0%	0%	96%	96%	100%
		3	8	0%	0%	0%	0%	8%	80%	88%	100%
			11	0%	0%	0%	0%	0%	68%	64%	100%
			14	0%	0%	0%	0%	0%	64%	60%	100%
		4	8	4%	0%	4%	8%	16%	68%	76%	100%
			11	0%	0%	0%	0%	4%	72%	76%	100%
			14	0%	0%	0%	0%	0%	36%	36%	96%
Overall				**7.9%**	**4.9%**	**8.0%**	**8.4%**	**22.6%**	**73.6%**	**75.1%**	**99.1%**

Table 6. Summary of results per parameter for *OB* experiment.

		OptFound			
Parameter		***Seed***	**SA_1**	**SA_2**	**SA***
p_{gen}	(1, 100)	38.2%	76.0%	77.3%	98.9%
	(100, 200)	6.9%	71.1%	72.9%	99.3%
d_{gen}	(1%, 5%)	22.4%	71.8%	74.2%	98.9%
	(5%, 10%)	22.7%	75.3%	76.0%	99.3%
m	2	22.3%	89.0%	90.3%	100.0%
	3	19.7%	69.0%	70.0%	99.7%
	4	25.7%	62.7%	65.0%	97.7%
n	8	37.3%	83.0%	86.0%	99.7%
	11	17.7%	70.7%	73.3%	99.0%
	14	12.7%	67.0%	66.0%	98.7%

Table 7. Percentage of times a list schedule finds the best seed for *BB* experiment.

p_{gen}	d_{gen}	m	n	$L(p_i^{min})$	$L(p_i^{max})$	$L(d_i^{min})$	$L(d_i^{max})$	$L(r_i^{min})$	$L(r_i^{max})$	$L(v_i^{min})$	$L(v_i^{max})$
(1, 100)	(1%, 5%)	4	20	24%	4%	8%	12%	20%	20%	28%	12%
			35	44%	8%	4%	4%	12%	4%	44%	16%
			50	44%	8%	4%	4%	4%	8%	48%	8%
		7	20	40%	12%	8%	4%	28%	16%	36%	12%
			35	36%	0%	4%	0%	40%	4%	32%	12%
			50	52%	4%	4%	0%	24%	0%	60%	4%
		10	20	56%	16%	8%	20%	48%	12%	52%	24%
			35	52%	8%	4%	8%	24%	8%	52%	0%
			50	40%	4%	4%	4%	40%	12%	28%	0%
	(5%, 10%)	4	20	32%	24%	0%	8%	40%	8%	32%	12%
			35	44%	4%	0%	4%	24%	4%	24%	16%
			50	24%	12%	4%	0%	24%	20%	16%	4%
		7	20	40%	12%	8%	4%	28%	20%	40%	16%
			35	20%	8%	4%	8%	32%	16%	24%	4%
			50	44%	4%	0%	0%	24%	12%	36%	0%
		10	20	64%	16%	24%	24%	40%	12%	60%	16%
			35	36%	4%	0%	8%	56%	4%	28%	0%
			50	40%	0%	4%	0%	32%	12%	32%	0%
(100, 200)	(1%, 5%)	4	20	24%	4%	12%	0%	28%	4%	28%	4%
			35	16%	8%	8%	12%	8%	8%	20%	20%
			50	20%	8%	4%	20%	4%	4%	12%	28%
		7	20	16%	24%	4%	0%	20%	12%	28%	4%
			35	16%	24%	4%	20%	4%	8%	16%	8%
			50	8%	12%	4%	12%	20%	8%	20%	16%
		10	20	12%	28%	8%	8%	12%	8%	12%	12%
			35	8%	12%	12%	24%	12%	0%	12%	20%
			50	20%	16%	12%	4%	16%	4%	16%	12%
	(5%, 10%)	4	20	12%	4%	12%	20%	8%	4%	20%	20%
			35	20%	20%	8%	8%	12%	12%	12%	8%
			50	28%	12%	4%	12%	32%	0%	8%	4%
		7	20	12%	20%	16%	4%	4%	16%	4%	24%
			35	0%	20%	8%	28%	16%	4%	16%	8%
			50	16%	8%	16%	0%	8%	4%	32%	16%
		10	20	8%	20%	12%	8%	4%	16%	16%	20%
			35	24%	4%	4%	4%	24%	4%	24%	12%
			50	0%	0%	24%	20%	12%	20%	16%	8%
Overall				**27.6%**	**10.9%**	**7.3%**	**8.8%**	**21.8%**	**9.1%**	**27.3%**	**11.1%**

Table 8. Average error and percentage of times the best solution is found for *BB* experiment.

				Error			*Best solutions found*		
p_{gen}	d_{gen}	***m***	***n***	**Seed**	**SA_1**	**SA_2**	**Seed**	**SA_1**	**SA_2**
(1, 100)	(1%, 5%)	4	20	6.4%	1.5%	1.6%	8%	36%	36%
			35	10.5%	2.8%	4.3%	0%	16%	4%
			50	13.8%	1.7%	2.8%	0%	12%	8%
		7	20	6.5%	2.8%	3.0%	20%	40%	40%
			35	11.2%	4.9%	5.2%	0%	16%	16%
			50	13.4%	4.1%	6.9%	0%	8%	0%
		10	20	5.6%	3.9%	2.7%	40%	40%	52%
			35	11.4%	6.4%	6.6%	4%	4%	4%
			50	14.1%	6.9%	6.9%	0%	0%	0%
	(5%, 10%)	4	20	8.3%	3.5%	3.4%	4%	36%	32%
			35	12.8%	2.6%	3.8%	0%	20%	12%
			50	14.2%	2.8%	3.4%	0%	8%	8%
		7	20	8.2%	3.6%	3.4%	20%	44%	44%
			35	13.3%	6.6%	6.5%	0%	12%	4%
			50	15.9%	6.3%	6.3%	0%	0%	0%
		10	20	6.4%	3.8%	3.3%	28%	60%	68%
			35	10.7%	7.6%	6.0%	4%	8%	12%
			50	15.8%	6.5%	8.6%	0%	12%	4%
(100, 200)	(1%, 5%)	4	20	9.6%	1.5%	1.9%	0%	28%	20%
			35	11.5%	0.9%	1.9%	0%	12%	0%
			50	12.3%	0.7%	1.4%	0%	12%	0%
		7	20	9.5%	2.9%	2.7%	0%	8%	16%
			35	14.7%	3.4%	4.4%	0%	16%	4%
			50	11.7%	1.9%	2.0%	0%	0%	12%
		10	20	13.4%	2.8%	2.8%	0%	8%	20%
			35	8.6%	2.2%	2.3%	0%	4%	4%
			50	16.5%	4.6%	6.4%	0%	4%	0%
	(5%, 10%)	4	20	9.6%	0.6%	1.3%	0%	44%	32%
			35	12.2%	0.9%	1.2%	0%	12%	4%
			50	15.2%	1.7%	2.7%	0%	4%	8%
		7	20	9.1%	3.0%	3.4%	0%	4%	4%
			35	15.8%	3.7%	3.4%	0%	12%	16%
			50	12.0%	2.1%	2.4%	0%	28%	8%
		10	20	13.6%	1.8%	1.9%	0%	24%	28%
			35	8.3%	1.8%	2.5%	0%	12%	4%
			50	17.9%	3.1%	4.1%	0%	4%	0%
Overall				**11.7%**	**3.3%**	**3.7%**	**3.56%**	**16.89%**	**14.56%**

Table 9. Summary of results per parameter for *BB* experiment

		Error				*Best Solutions Found*		
Parameter		**Seed**	**SA_1**	**SA_2**		**Seed**	**SA_1**	**SA_2**
p_{gen}	(1, 100)	11.03%	4.36%	4.70%		7.11%	20.67%	19.11%
	(100, 200)	12.31%	2.20%	2.69%		0.00%	13.11%	10.00%
d_{gen}	(1%, 5%)	11.15%	3.12%	3.64%		4.00%	14.67%	13.11%
	(5%, 10%)	12.20%	3.44%	3.74%		3.11%	19.11%	16.00%
m	4	11.37%	1.78%	2.46%		1.00%	20.00%	13.67%
	7	11.77%	3.77%	4.12%		3.33%	15.67%	13.67%
	10	11.87%	4.29%	4.50%		6.33%	15.00%	16.33%
n	20	8.85%	2.63%	2.60%		10.00%	31.00%	32.67%
	35	11.75%	3.67%	4.00%		0.67%	12.00%	7.00%
	50	14.41%	3.54%	4.48%		0.00%	7.67%	4.00%

Table 10. CPU times (in seconds) per parameter for *BB* experiment

		Error		
Parameter		**SA_1**	**SA_2**	**SA***
p_{gen}	(1, 100)	4.3	4.2	415.5
	(100, 200)	4.8	4.7	466.8
d_{gen}	(1%, 5%)	4.5	4.5	433.9
	(5%, 10%)	4.6	4.4	448.4
m	4	8.6	8.5	837.1
	7	3.2	3.2	308.0
	10	1.8	1.7	178.3
n	20	0.2	0.2	24.2
	35	2.4	2.4	238.5
	50	11.0	10.8	1,060.7
Overall		4.54	4.46	441.1

Figure 1. Example problem schedules.