

Heuristics for the Unrelated Parallel Machine Scheduling Problem with Setup Times

Ghaith Rabadi*, Dept. of Engineering Management and Systems Engineering, Old Dominion University, Norfolk, Virginia 23523, USA, Email: grabadi@odu.edu, Phone: 757-683-4918, Fax: 757-683-5640

Reinaldo J. Moraga, Dept. of Industrial Engineering, Universidad del Bío Bío, Concepción, Chile

Ameer Al-Salem, Dept. of Mechanical Engineering, University of Qatar, Doha, Qatar

Abstract

The problem addressed in this paper is the non-preemptive unrelated parallel machine scheduling problem with the objective of minimizing the makespan. Machine-dependent and job sequence-dependent setup times are considered, all jobs are available at time zero, and all times are deterministic. This is a NP-hard problem and in this paper, optimal solutions are found for small problems only. For larger problems, a new meta-heuristic, Meta-RaPS, is introduced and its performance is evaluated by comparing its solutions to the solutions of an existing heuristic for the same problem. The results show that Meta-RaPS found all optimal solutions for the small problems and outperformed the solutions obtained by the existing heuristic for larger problems.

* Corresponding author

1. Introduction

The unrelated parallel machines scheduling problem (PMSP) addressed in this paper is the scheduling of n jobs that are available at time zero on m unrelated machines (R_m) to minimize the makespan, C_{max} , without preemption. Machines are considered unrelated when jobs' processing times depend on the machine to which they are assigned, and when there is no relationship between machine speeds. Machine-dependent sequence-dependent setup times S_{ijk} are also considered. The setup times are sequence-dependent as their amounts depend on job sequence. They are also machine-dependent in a sense that each machine has its own matrix of setup times. The problem addressed here will be referred to hereafter as $R_m|S_{ijk}|C_{max}$. The basic identical PMSP $P_m||C_{max}$ is NP-hard even when $m = 2$ (Karp, 1972; Garey and Johnson, 1979). Since $R_m|S_{ijk}|C_{max}$ is a generalization of the former problem, then it is also NP-hard.

Having multiple resources with different capabilities in parallel is common in many industries to obtain adequate capacity. In addition, setup times are usually incurred when switching from one item to another. Applications for parallel machine scheduling with setup are common in many industries including painting, plastic, textile, glass, semiconductor, chemical, and paper manufacturing as well as some service industries (e.g., Guinet, 1991; Franca et al., 1996; Radhakrishnan and Ventura, 2000; Kurz and Askin, 2001; Randhawa and Kuo, 2001). The motivation for this paper comes from the truss manufacturing industry where manufacturing roof trusses requires different setup times depending on the manufacturing sequence and on the type of manufacturing apparatus used. A truss includes m joints and n components where the truss design accurately specifies the length and cut angles on all of the components and the location of

joints where the components are fastened. The manufacturing process requires preparing components by cutting, assembling, and fastening them into a particular truss using connector plates. Each truss requires a particular setup that is used for placing the components at appropriate joint locations in a layout jig. The setup time for a given truss depends on the type of machine on which the truss will be laid out and the truss components joined. Due to the complexity of the problem, finding optimal solutions for large problems is very time consuming and sometimes computationally infeasible. Developing heuristic algorithms to reach near-optimal solutions becomes much more practical and useful.

Although the literature on parallel machines problems is still not as generous as in the case of single-machine scheduling problems, a growing research activity is definitely noted starting from the early McNaughton's (1959) initial work. State-of-art reviews on parallel machines' research in general can be found in Graves (1981), Cheng and Sin (1990), Lawler *et al.* (1993), and more recently in Mokotoff (2001). In addition, Allahverdi *et al.* (1999) presented a survey on scheduling problems involving setup times constraints. While the focus of our literature review will be limited to the unrelated PMSP, it is worth mentioning that many papers addressed the *identical* parallel machine scheduling with and without setup consideration including most recently Dunstall and Wirth (2005), Kurz and Askin (2001), and Lin and Wenhua (2004).

The unrelated PMSP has been far less studied than other environments (Dhaenens-Flipo, 2001). Bruno *et al.* (1974) and Horn (1973) had examined the unrelated PMSP to minimize the total completion time, while Azizoglu and Kirca (1999) have studied the same problem but considering the total weighed completion time. Hariri and Potts (1991)

introduced a two-phase heuristic approach in which they applied linear programming in its first phase to generate a partial schedule, then they used the earliest completion time heuristic in the second phase to schedule the remaining jobs to minimize the makespan. Weng *et al.* (2001) addressed the unrelated PMSP to minimize the weighted mean completion time and included machine-independent sequence-dependent setup times. They presented and tested seven heuristics in which they either assigned a job to the machine with the least cost contribution, or to the machine on which the job has the shortest processing time. They also tried a strategy where they assigned first the job with the smallest ratio of processing time plus setup time to weight. This strategy outperformed the rest significantly. The authors reported that their algorithms are fast and solved problems up to 120 jobs and 12 machines. Bank and Werner (2001) considered the same problem with release dates, but their objective was to minimize the weighted sum of earliness and tardiness. They compared several constructive and iterative algorithms and concluded that their approach performed well.

Some authors have considered the use of metaheuristic approaches for the unrelated PMSP. Kim *et al.* (2002, 2003) developed heuristics for the problem including a Simulated Annealing (SA) to minimize the total tardiness with machine-independent sequence-dependent setup times. Glass *et al.* (1994) compared genetic algorithms (GA), SA, and Tabu Search (TS) for $R_m||C_{max}$ without setup times. The authors concluded that the quality of solutions generated by the GA was poor. However, a hybrid method that incorporated a GA was comparable to performances by SA and TS. Srivastava (1997) presented an effective TS for the same problem without setup times and reported that the TS can provide good quality solutions for practical size problems within a reasonable

amount of time. Ghirardi and Potts (2005) also addressed $R_m||C_{max}$ where the heuristic they used was an application of the Recovering Beam Search. The authors reported that their algorithm produced good results on large instances (up to 50 machines and 1000 jobs).

Some researchers developed exact algorithms for the unrelated PMSP including Liaw *et al.* (2003) and Lancia (2000) who developed branch-and-bound algorithms to find optimal solutions for the problem without setup times, where the objective functions were the total weighted tardiness, and C_{max} respectively. Martello *et al* (1997) developed lower bounds for $R_m||C_{max}$ based on Lagrangian relaxation, which showed to be better than previous bounds. They utilized their results also to develop effective approximate algorithms.

Based on our previous literature review, most of the research that addressed the unrelated PMSP either did not consider sequence-dependent machine-dependent setup times and/or dealt with other objective functions except for Al-Salem (2004) who addressed the same problem. In this paper, a new heuristic algorithm is introduced to solve this problem, and the results obtained will be compared to those obtained by the Partitioning Heuristic introduced by Al-Salem (2004).

While the PMSP has been addressed in the literature due to its important applications, benchmark data sets for the unrelated PMSP with setups are unavailable except for those used by Al-Salem (2004). Therefore, the data sets used in this paper along with the solutions obtained by Partitioning Heuristic as well as the proposed new heuristic are available in Rabadi (2005).

2. Optimal Solutions Using Integer Programming

A Mixed Integer Program (MIP) is formulated to find optimal solutions for the problem at hand. Similar formulation was used by Guinet (1991).

$$\text{Minimize } C_{\max} \quad (1)$$

subject to

$$\sum_{\substack{i=0 \\ i \neq j}}^n \sum_{k=1}^m x_{i,j,k} = 1 \quad \forall j = 1, \dots, n \quad (2)$$

$$\sum_{\substack{i=0 \\ i \neq h}}^n x_{i,h,k} - \sum_{\substack{j=0 \\ j \neq h}}^n x_{h,j,k} = 0 \quad \forall h = 1, \dots, n, \forall k = 1, \dots, m \quad (3)$$

$$C_j \geq C_i + \sum_{k=1}^m x_{i,j,k} (S_{i,j,k} + p_{j,k}) + M \left(\sum_{k=1}^m x_{i,j,k} - 1 \right) \quad \forall i = 0, \dots, n, \forall j = 1, \dots, n, \quad (4)$$

$$\sum_{j=0}^n x_{0,j,k} = 1 \quad \forall k = 1, \dots, m \quad (5)$$

$$x_{i,j,k} \in \{0,1\} \quad \forall i = 0, \dots, n, \forall j = 0, \dots, n, \forall k = 1, \dots, m \quad (6)$$

$$C_0 = 0 \quad (7)$$

$$C_j \geq 0 \quad \forall j = 1, \dots, n \quad (8)$$

where,

C_j : Completion time of job j

$p_{j,k}$: Processing time of job j on machine k

$S_{i,j,k}$: Sequence-dependent setup time to process job j after job i on machine k

$S_{0,j,k}$: Setup time to process job j first on machine k

$x_{i,j,k}$: 1 if job j is processed directly after job i on machine k and 0 otherwise

$x_{0,j,k}$: 1 if job j is the first job to be processed on machine k and 0 otherwise

$x_{j,0,k}$: 1 if job j is the last job to be processed on machine k and 0 otherwise

M : a large positive number

The objective (1) is to minimize the makespan. Constraints (2) ensure that each job is scheduled only once and processed by one machine. Constraints (3) make sure that each job must neither be preceded nor succeeded by more than one job. Constraints (4) are used to calculate completion times and to ensure that no job can precede and succeed the same job. Constraints (5) ensure that no more than one job can be scheduled first at each machine. Note that there is no need for another set of constraints to guarantee that only one is scheduled last on each machine because this is guaranteed by constraints (5) in conjunction with (3). Constraints (6) specify that the decision variable x is binary over all domains. Constraints (7) state that the completion time for the dummy job 0 is zero and constraints (8) ensure that completion times are non-negative. Optimal solutions for the problem can be obtained by solving the previous MIP using a solver.

3. Meta-Raps Overview and its application to $R_m/S_{ijk}/C_{max}$

Meta-RaPS (Meta-heuristic for Randomized Priority Search) is a master strategy that uses both construction and improvement heuristics to generate high quality solutions. Meta-RaPS is the result of research conducted on the application of a modified COMSOAL approach to several combinatorial problems. COMSOAL (Computer Method of Sequencing Operations for Assembly Lines) is a computer heuristic originally

reported as a solution approach to the assembly line balancing problem (Arcus, 1966). Despite the fact that Meta-RaPS conserves in essence Arcus's underlying original COMSOAL idea, in practice, the methods differ considerably. Meta-RaPS has been successfully applied to a number of combinatorial problems such as the Resource Constrained Project Scheduling Problem (DePuy and Whitehouse, 2001), the Traveling Salesman Problem (DePuy *et al.* 2004), the 0-1 Multidimensional Knapsack Problem (Moraga *et al.*, 2004), and the Vehicle Routing Problem (Moraga, 2002).

In Meta-RaPS, as with other meta-heuristics, randomness is a mechanism used to prevent the algorithm from getting trapped into local optima. Its performance is controlled by using four parameters; the number of iterations (I), the priority percentage ($\%p$), the restriction percentage ($\%r$), and the improvement percentage ($\%i$). During Meta-RaPS execution, the number of iterations determines the number of feasible solutions constructed. In general, a construction heuristic builds a solution by systematically adding feasible jobs to the current schedule. The job with the best priority value is added to the current schedule. Meta-RaPS modifies the way a general construction heuristic chooses the next job to add to the solution by occasionally choosing a job that does not have the best priority value. In the construction stage, the $\%p$ parameter is used to determine the percentage of time the next item is added to the solution by using the best priority value. In the remaining time ($100\%-\%p$), the next item added to the solution is randomly chosen from those feasible items whose priority values are within the best $\%r$ of the best priority value. Proceeding in this way, Meta-RaPS introduces diversification to the process while maintaining the same quality produced by the priority rule. In addition, a solution improvement algorithm can be included in Meta-

RaPS and the $\%i$ parameter is used. Once a schedule has been constructed, Meta-RaPS may proceed to further improve it through neighborhood search algorithms. The improvement heuristic is performed if the construction stage's solution value is within the lowest $\%i$ of the range between both best and worst unimproved solution values found so far. Finally, the best solution from all iterations is reported.

Meta-RaPS can be considered a general form of COMSOAL. Meta-RaPS with parameters of $\%p=0$, an infinitely large $\%r$, and $\%i=0$ will mimic COMSOAL. Meta-RaPS is also the general form of another meta-heuristic called GRASP (Greedy Randomized Adaptive Search Procedure; Feo and Resende, 1995). GRASP constructs solutions by introducing randomness to a greedy construction heuristic through the use of a restriction percentage parameter similar to Meta-RaPS, but it does not give any probabilistic priority to the best alternative considered by the greedy construction heuristic. GRASP also includes a local search improvement heuristic applied to all constructed solutions. GRASP has been applied to a variety of scheduling problems (Feo *et al.*, 1991; Laguna and González-Velaverde, 1991; Feo *et al.*, 1996; Aiex *et al.*, 2003; and Rojanasoonthon and Bard, 2004); however no research for the $R_m|S_{ijk}|C_{max}$ using GRASP has been done to date. Using Meta-RaPS with parameters of $\%p=0$, and $\%i=100$ will imitate GRASP. Meta-RaPS offers greater flexibility over COMSOAL and GRASP in that it allows user-defined settings for the three parameters $\%p$, $\%r$, and $\%i$. DePuy *et al.* (2005) show that the extra flexibility of Meta-RaPS seems beneficial by demonstrating that the Meta-RaPS' parameter settings used to find the best solutions for the traveling salesman problem (TSP) are different from those settings used to imitate COMSOAL or GRASP. The general Meta-RaPS procedure to solve the $R_m|S_{ijk}|C_{max}$ as

well as the construction and improvement heuristics used are explained in more detail in the following sections.

3.1 Meta-RaPS Construction Heuristic for $R_m|S_{ijk}/C_{max}$

Let $[AP]_k$ denote the *Adjusted Processing Time* matrix for each machine k , such that: $AP_{ijk} = S_{ijk} + P_{jk}$, $\forall i=0, \dots, n$, $\forall j=1, \dots, n$, and $\forall k=1, \dots, m$; where the first row in $[AP]_k$ contains the initial setup times for each job at each machine k . A similar approach of merging the processing and setup times in one matrix was used by Gendreau *et al.* (2001) and Rabadi *et al.* (2004). The entry (i, j, k) in the matrix $[AP]_k$ is read as job j scheduled immediately after job i on machine k . In addition, let $l(k)$ and L_k denote the last scheduled job and the load on machine k , respectively. The rationale of the construction is to schedule first the job with the **Shortest Adjusted Processing** time to the machine with the **Smallest Load** (thus the name SAP-SL). The SAP-SL algorithm is described as follows:

SAP-SL Algorithm for $R_m|S_{ijk}|C_{max}$

Step 0. Let S be a schedule in the format $\{\text{jobs on machine1}; \text{jobs on machine2}; \dots\}$, and let M_0 denote the set of machines $\{1, \dots, m\}$.

Set $S = \emptyset$, and $J^a = \{1, \dots, n\}$. Initially, $l(k) = 0$, $pos(k) = 0$, and $L_k = 0$ for all $k \in M_0$.

Step 1. Let $M = \{all\ k \in M_0 \mid L_k^* = \min_{k \in M_0} (L_k)\}$, where L_k^* is the minimum load.

Step 2. Let j^* and k^* denote the job and machine that respectively satisfy

$$\alpha_{j^*k^*} = \min_{\substack{k \in M \\ j \in J^a}} \{AP_{ijk}\},$$

ties are broken arbitrarily.

Step 3. Increase $pos(k^*)$ by one. Assign job j^* to $S(k^*)$ in $pos(k^*)$.

Step 4. For machine k^*

$$L_{k^*} = L_k^* + \alpha_{j^*k^*}$$

Eliminate column j^* and row $l(k^*)$ from $[AP]_{k^*}$, $l(k^*)=j^*$.

For the rest of machines

Eliminate row and column j^* .

Delete j^* from J^a .

Step 5. If $J^a = \emptyset$ STOP and compute $C_{max}(S) = \max_{k \in M_0} \{L_k\}$, otherwise go to Step 1.

To demonstrate how SAP-SL works, consider an example of two machines ($k = 2$) and six jobs ($n = 6$) with $[AP]_1$ and $[AP]_2$ as shown in Table 1 (the entry $AP_{3,5,2} = 127$ for example, indicates the adjusted processing time of job 5 after job 3 in machine 2). The first two assignments are shown in detail as follows:

Table 1. Initial data

AP ₁	1	2	3	4	5	6		AP ₂	1	2	3	4	5	6
0	116	142	130	109	157	152		0	163	135	149	136	127	131
1	--	167	166	122	179	141		1	--	135	171	126	139	156
2	127	--	120	145	170	180		2	144	--	128	158	165	110
3	143	165	--	150	143	145		3	156	154	--	145	127	136
4	124	165	157	--	173	137		4	147	152	159	--	170	140
5	118	162	158	108	--	137		5	144	142	168	171	--	127
6	132	170	136	151	181	--		6	166	157	172	172	173	--

First assignment:

Step 0. $S=\emptyset, J^a=\{1, 2, 3, 4, 5, 6\}, M_0 = \{1, 2\}, l(1)=l(2)=0, pos(1)=pos(2)=0,$
and $L_1= L_2=0.$

Step 1. $M=\{1, 2\},$ since the minimum load is $L_{1,2}^* = 0$ for machines 1 and 2.

Step 2.

$$\alpha_{j^*k^*} = \min_{\substack{k \in \{1,2\} \\ j \in J^a}} \{116,142,130,109,157,152; \\ 163,135,149,136,127,131\} = 109, \text{ for } j^* = 4 \text{ and } k^* = 1.$$

(See highlighted cell in Table 1)

Step 3. $pos(1)=0+1.$ Assign $S(1)=4$ in $pos(1)=1.$

Step 4. For machine $k^*=1, L_1=0+109=109;$ eliminate column $j^*=4$ and row $l(1)=0$ from $[AP]_1,$ and set $l(1)=4.$ For the rest of machines, eliminate row and column $j^*=4;$ as shown in Table 2. Delete $j^*=4$ from $J^a.$ The new set $J^a = \{1,2,3,5,6\}.$

Table 2. Data after first job assignment

AP ₁	1	2	3	4	5	6		AP ₂	1	2	3	4	5	6
0	--	--	--	--	--	--		0	163	135	149	--	127	131
1	--	167	166	--	179	141		1	--	135	171	--	139	156
2	127	--	120	--	170	180		2	144	--	128	--	165	110
3	143	165	--	--	143	145		3	156	154	--	--	127	136
4	124	165	157	--	173	137		4	--	--	--	--	--	--
5	118	162	158	--	--	137		5	144	142	168	--	--	127
6	132	170	136	--	181	--		6	166	157	172	--	173	--

Step 5. Since $J^a \neq \emptyset$, go to Step 1 for the second assignment.

Second assignment:

Step 1. $M=\{2\}$, since the minimum load is $L_2^* = \min\{109, 0\} = 0$ for machine 2.

Step 2.

$$\alpha_{j^*k^*} = \min_{\substack{k \in \{2\} \\ j \in J^a}} \{163, 135, 149, \dots, 127, 131\} = 127, \text{ for } j^* = 5 \text{ and } k^* = 2.$$

(See highlighted cell in Table 2)

Step 3. $pos(2)=0+1$. Assign $S(2)=5$ in $pos(2)=1$.

Step 4. For machine $k^*=2$, $L_2=0+127=127$; eliminate column $j^*=5$ and row $l(2)=0$ from $[AP]_2$, and set $l(2)=5$. For the rest of machines, eliminate row and column $j^*=5$; as shown in Table 3. Delete $j^*=5$ from J^a . The new set $J^a = \{1, 2, 3, 6\}$.

Table 3. Data after second job assignment

AP ₁	1	2	3	4	5	6		AP ₂	1	2	3	4	5	6
0	--	--	--	--	--	--		0	--	--	--	--	--	--
1	--	167	166	--	--	141		1	--	135	171	--	--	156
2	127	--	120	--	--	180		2	144	--	128	--	--	110
3	143	165	--	--	--	145		3	156	154	--	--	--	136
4	124	165	157	--	--	137		4	--	--	--	--	--	--
5	--	--	--	--	--	--		5	144	142	168	--	--	127
6	132	170	136	--	--	--		6	166	157	172	--	--	--

Step 5. Since $J^a \neq \emptyset$, go to Step 1 for the third assignment.

Proceeding in the described way, the third assignment schedules job 1 after job 4, on machine 1. The fourth assignment schedules job 6 after job 5 on machine 2. The fifth assignment schedules job 3 on machine 1, and the sixth assignment schedules job 2 on machine 2. The final sequence, S , is $\{4\ 1\ 3; 5\ 6\ 2\}$ with a makespan, $C_{max}(S)$, of 411.

Meta-RaPS uses SAP-SL as its basic construction procedure. However, while adding jobs to the solution, the $\%p$ parameter is used to determine the percentage of time the next job added to the solution has the best priority value ($\alpha_{j^*k^*}$) as in Step 2. The remaining time $(100\% - \%p)$, the next job added to the solution is randomly chosen from those feasible jobs whose AP_{ijk} values are within $\%r$ above the $\alpha_{j^*k^*}$ value, which further modifies Step 2. Therefore, Step 2 becomes Step 2' as follows:

Step 2'. *Generate a random number rnd .*

If $rnd \leq \%p/100$,

Then, j^ and k^* are the job and machine that respectively satisfy*

$$\alpha_{j^*k^*} = \min_{\substack{k \in M \\ j \in J^a}} \{AP_{ijk}\},$$

Else, take j^ and k^* randomly from the set of all $j \in J^a$ and $k \in M$ that satisfy*

$$AP_{ijk} \leq \alpha + (\beta - \alpha)\%r/100,$$

$$\text{where } \alpha = \min_{\substack{k \in M \\ j \in J^a}} \{AP_{ijk}\}, \beta = \max_{\substack{k \in M \\ j \in J^a}} \{AP_{ijk}\}$$

The rationale in Step 2' takes advantage of two experimental findings. First, if some degree of randomness is introduced into a particular heuristic (or priority rule), by using $\%r$, the corresponding results improve dramatically. Second, if two heuristics are randomly combined, by using $\%p$, the solution obtained usually outperforms the solution of either heuristic individually (DePuy *et al.*, 2005).

If parameters $\%p=20$ and $\%r=40$ are used in the previous example, for the first assignment, Steps 0 and 1 remain the same as before. However, in Step 2', Meta-RaPS generates a random number, say $rnd_1=0.25$. Since $rnd_1 \geq 0.20$, j^* and k^* are randomly taken from the set of all the jobs $j \in \{1,2,3,4,5,6\}$ and machines $k \in \{1,2\}$ that satisfy the inequality $AP_{jk} \leq [109 + (163 - 109) * 0.40] = 130.6$. Suppose a random selection of $j=3$ and $k=1$, then $AP_{031}=130$, which is less than 130.6; thus $j^*=3$ and $k^*=1$ are accepted and the subsequent Steps 3, 4, and 5 proceed in the same manner as in the previous example. After all assignments are made through Meta-RaPS' constructive procedure, the schedule $S=\{3\ 5\ 4; 1\ 2\ 6\}$ with a $C_{max}(S) = 408$ is obtained.

This method of random selection is thought to decrease the possibility of being trapped in local optima, and to find an answer closer to the global optimum. Next, an improvement heuristic is applied on those constructed solutions that are within the lowest $\%i$ of the range between both best and worst unimproved makespans.

3.2 Meta-RaPS Improvement Heuristic for $R_m/S_{ijk}/C_{max}$

Once a schedule S has been constructed, the $\%i$ parameter is used to decide whether S will pass through an improvement stage or not. While Meta-RaPS is being executed, it keeps track of the best and worst makespans of schedules constructed so far. If $C_{max}(S)$,

is within the lowest $\%i$ of the range between the best and worst makespans, S will go through the improvement stage. The improvement in Meta-RaPS consists of three well-known local search techniques: inter-machine job insertion, inter-machine pairwise job exchange, and intra-machine random job exchange, which are performed on a cyclical basis. While inter-machine transitions exchange/insert jobs across machines, the intra-machine transitions exchange jobs on the same machine (see Pinedo (2002) for more detail). Each technique uses a best-improved-solution approach. After the three techniques are applied during ten cycles, the improvement stage keeps the best schedule found. The number of cycles was chosen by keeping a tradeoff between solution quality and computational cost of the improvement stage. For the example, if the parameter $\%i=50$ and the current best and worst constructed makespans are 395 and 452 respectively; then, since the solution obtained by Meta-RaPS, 408, is less than 424 ($=395+(452-395)0.50$), the solution goes through the improvement stage. The final solution is a makespan of 395, which coincides with the optimum for this small problem instance. The improvement stage procedure is shown in pseudo-code as follows:

If $C_{max}(S) \leq Best + (Worst - Best)\%i/100$

Cycles=10; *count*=0; $S=S'$;

While $count \leq Cycles$

count=*count*+1;

$C_{max}(S_1) = C_{max}(\textit{insertion on } S')$;

$C_{max}(S_2) = C_{max}(\textit{inter_machine_exch on } S')$;

$C_{max}(S_3) = C_{max}(\textit{intra_machine_exch on } S')$;

Find S' corresponding to $\min\{ C_{max}(S_1), C_{max}(S_2), C_{max}(S_3) \}$;

End;

S=S';

4. The Partitioning Heuristic

The Partitioning Heuristic algorithm developed by Al-Salem (2004) for $R_m|S_{ijk}|C_{max}$ applies three phases sequentially. The first phase is a constructive heuristic to assign jobs to machines. The second phase is an improvement heuristic applied to the solution provided by the constructive phase. Finally, the third phase is a heuristic that deals with each machine as a TSP and determines the sequence of jobs on each machine.

To initially assign jobs to machines in phase 1, the processing time plus the average setup times for each job on each machine is computed and the ratio of the minimum processing time plus the average machine setup times is also determined. If the ratio is small enough (e.g., < say 0.7), the job is assigned to the machine with shortest processing time plus the average setup times; otherwise the job is considered pending. Pending jobs are then arranged by decreasing order of the average processing times plus average setup times. Following that order, pending jobs are assigned to machines that result in the lowest partial *estimated* makespan. The value of the estimated makespan \hat{C}_{max} is obtained similar to the method used by Lee *et al.* (1997) to estimate the value of the makespan for the single machine scheduling problem with sequence-dependent setup times. Therefore $\hat{C}_{max} = (\theta\bar{s} + \bar{p})n$ where θ is a coefficient that takes into account the

effect of the setup times on the makespan. The average time to process a job including its setup time is assumed to be $\hat{C}_{\max} = \theta \bar{s} + \bar{p}$ with $\theta \leq 1$.

The result of phase 1 is an assignment of all jobs to machines. At this point, \hat{C}_{\max} is available for each machine, but not the actual makespan and the jobs have not been re-sequenced to reduce the makespan. To reduce the makespan, the improvement phase is applied to the schedule generated in phase 1. In this application, the Composite Exchange Heuristic (CEH) developed by Hariri and Potts (1991) is modified to account for the sequence-dependent setup times and applied to the schedule from phase 1. The modified CEH consists of two stages: in the first stage, a job is removed from the machine that produces the largest \hat{C}_{\max} and is assigned to a machine that produces the lowest \hat{C}_{\max} . All jobs and all possible assignments are considered in the second stage; two jobs are exchanged, one from the machine that produces the largest \hat{C}_{\max} and one from the machine that produces the lowest \hat{C}_{\max} . The first stage is applied by first using the constructive schedule as an input. Then using the resulting improved schedule as an input to the second stage, a further reduction in \hat{C}_{\max} is attempted. The procedure continues by repeatedly applying the first stage then the second stage until no further reduction in makespan is possible.

To find the sequence and the actual makespan on each machine, phase 3 is applied where each machine is treated as a TSP. Two neighborhood heuristics are used to solve the related TSP on each machine: the Nearest Neighbor Heuristic (NNH) is used first to obtain the initial sequence followed by the Adjacent Pairwise Interchange heuristic (API) to improve the makespan on the machine that produces the largest makespan. NNH starts

processing the job that has the smallest processing time plus setup time, then, whenever a job is completed, the job with the smallest processing time plus setup time is selected to go next. The API then swaps a job with the job that directly precedes or succeeds it in the schedule. This procedure is repeated for all adjacent jobs. See Al-Salem (2004) for more detail on the Partitioning Heuristic.

5. Computational Experience

5.1 Meta-RaPS Parameter Setting

Parameters in Meta-RaPS can be set either arbitrarily or by using a more systematic method. In this case, the following general parameter setting framework (Moraga *et al.*, 2005) was used:

Step 1. *Select a representative subset of problems to analyze from the entire collection of problems.* In this study, 15 problem instances were randomly generated for each pair (m, n) . A subset of two instances was randomly selected for each pair.

Step 2. *Select the parameter domain over which each parameter will be varied.* The parameter domain was varied over the whole range (0-100%) for $\%p$ and $\%r$, and over 40-80% range for $\%i$. An increment size of 10% was used for parameter $\%p$. For parameters $\%r$ and $\%i$, an increment size of 5% and 10% were used respectively. The number of iterations, I , was fixed to 5000.

Step 3. *For each problem in the subset, run Meta-RaPS using an appropriate technique for setting parameters.* In this step, the technique consisted of sequentially varying each parameter over its domain until a value was found while keeping other values fixed. First, the $\%r$ parameter was varied while fixing $\%p$ and $\%i$ to

zero. The $\%r$ parameter associated with the best solution value was used. Second, the $\%p$ parameter was varied while maintaining $\%r$ at its setting and $\%i$ at zero. Finally, the $\%i$ parameter is varied while keeping $\%p$ and $\%r$ at their settings.

Step 4. *Use the overall best parameter settings obtained in Step 3 for the entire collection of problems.* In this case, the 15 instances, and repeat the process for each pair (m, n) .

It should be noted that the parameter selection methodology presented in this section may not produce the optimal combination of parameter values. The intention is to provide a systematic method of selecting good parameters.

5.2 Results

Two experiments were designed to test the effectiveness of the Meta-RaPS algorithm. In the first experiment, small instances were solved using both the proposed Meta-RaPS and the existing Partitioning Heuristic. Both algorithms were compared to optimal solutions of small-sized instances. Optimal solutions were obtained using OPL (Optimization Programming Language), which utilizes CPLEX 7.0 as a solver (see Van Hentenryck (2001) for more detail on OPL). In the second experiment, large problem instances were generated and solved using both Meta-RaPS and the Partitioning Heuristic. Optimal solutions were unattainable for large instances, and therefore, solutions from both heuristics were compared to each other. In both experiments, the processing and setup times were randomly drawn from two uniform distributions: $U[50, 100]$ and $U[125, 175]$. Uniform distribution is widely used for this purpose because

many real world instances match such data settings, and it has high variance, which allows the proposed heuristics to be tested under unfavorable conditions (Weng *et al.*, 2001). The selection of the uniform distribution bounds for processing and setup times determines the level of dominance. That is, when processing times and setup times are balanced (denoted by P_{ij}, S_{ij} *Balanced*), they are both drawn from U[50, 100]. When processing times are dominant (denoted by P_{ij} *Dominant*), the processing times and the setup times are drawn from U[125, 175] and U[50, 100] respectively. When the setup times are dominant (denoted as S_{ij} *Dominant*), the processing times and the setup times are drawn from [50, 100] and [125, 175] respectively. Both heuristics were implemented in C++ and the experiments were run on a Pentium IV- 1.7 GHz personal computer.

Experiment 1

Problem instances ranging from 6 to 9 jobs and 2 to 4 machines were randomly generated and solved optimally. In case of 4 machines, optimal solutions for a maximum of 8 jobs were found in a feasible time. For each combination of machine number, job number, processing time distribution, and setup time distribution, 15 instances were solved. This resulted in a total of 630 problem instances that were solved by each algorithm. The percentage deviation (ρ) of each heuristic solution value above the optimal solution value was used as a measure of performance for each problem instance. That is:

$$\rho = \frac{C_{\max}(\text{Heuristic}) - C_{\max}(\text{Optimal})}{C_{\max}(\text{Optimal})} \times 100$$

Meta-RaPS was able to reach optimal solutions for all instances (i.e., $\rho = \text{zero}$ for all instances) regardless whether the processing or setup times were dominant or not. Table 4 shows the ρ values for the Partitioning Heuristic.

Table 4 . Average ρ Values of the Partitioning Heuristic for Small Problems

m	n	$P_{ij}, S_{ij} \text{ Balanced}$	$P_{ij} \text{ Dominant}$	$S_{ij} \text{ Dominant}$
2	6	3.70	1.69	1.97
	7	4.90	3.19	2.18
	8	3.30	2.67	2.67
	9	5.69	2.57	2.85
4	6	3.30	2.35	2.86
	7	4.22	3.44	3.41
	8	2.79	2.12	2.31

Experiment 2

Large test problems ranging from 20 to 120 jobs, and 2 to 12 machines were generated. 15 instances were solved for each combination of machine number, job number, processing time distribution, and setup time distribution. This resulted in a total of $6 \times 6 \times 3 \times 15 = 1620$ problem instances that were solved by each algorithm. The percentage deviation of the Partitioning Heuristic solution values above the Meta-RaPS solution values (δ) was used as a measure of performance for each problem instance.

That is:

$$\delta = \frac{C_{\max}(\text{Partitioning Heuristic}) - C_{\max}(\text{MetaRaps})}{C_{\max}(\text{MetaRaps})} \times 100$$

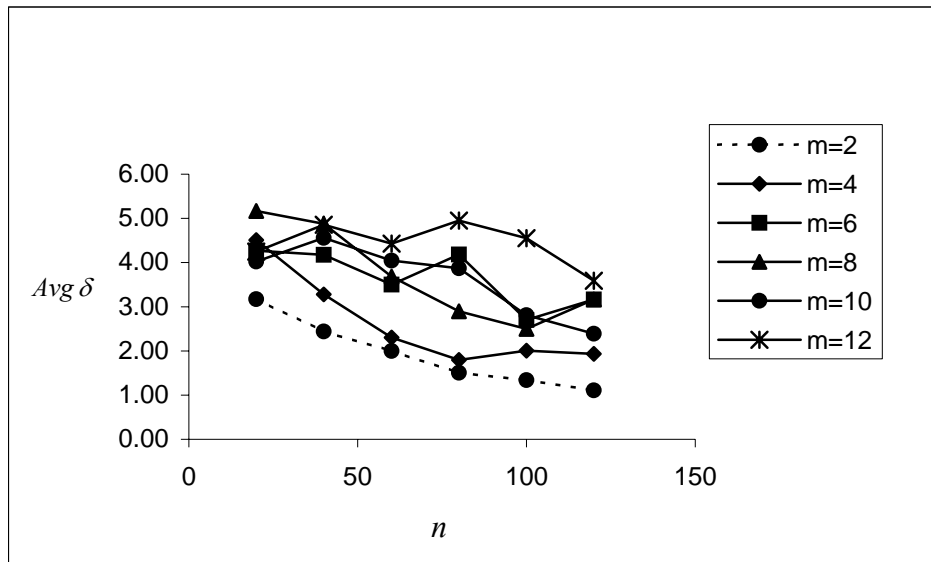
Table 5 shows the average values of δ when the processing and setup times are balanced, when the processing times are dominant, and when the setup times are

dominant. It is obvious that the Meta-RaPS algorithm outperformed the existing Partitioning Heuristic.

Table 5. Average δ Values of the Partitioning Heuristic for Large Problems

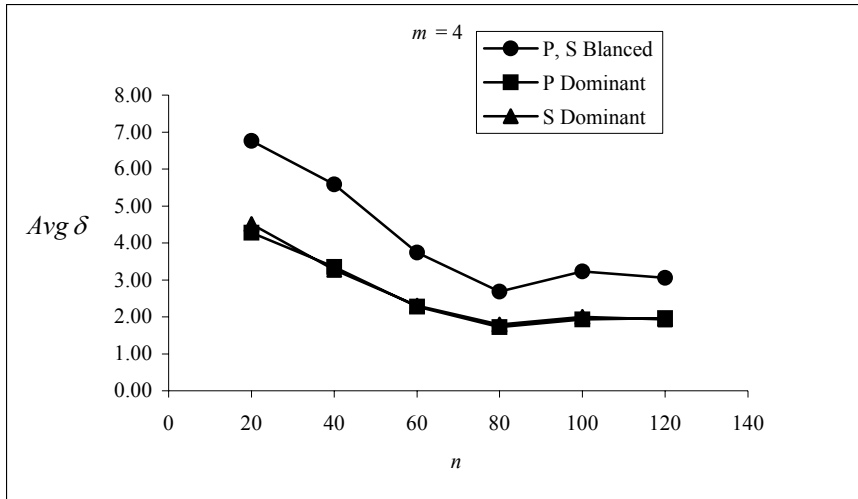
m	n	$P_{ij}, S_{ij} \text{ Balanced}$	$P_{ij}, \text{ Dominant}$	$S_{ij} \text{ Dominant}$
2	20	4.70	3.17	3.11
	40	4.10	2.44	2.88
	60	3.19	2.00	1.88
	80	2.57	1.50	1.58
	100	2.33	1.34	1.22
	120	1.95	1.11	1.04
4	20	6.76	4.27	4.51
	40	5.59	3.35	3.28
	60	3.74	2.27	2.30
	80	2.68	1.72	1.80
	100	3.23	1.93	2.00
	120	3.05	1.96	1.93
6	20	7.76	4.54	4.27
	40	6.90	3.99	4.18
	60	6.96	3.24	3.51
	80	6.05	4.23	4.18
	100	4.27	2.92	2.70
	120	4.49	3.12	3.16
8	20	7.34	3.89	5.17
	40	7.70	4.75	4.88
	60	8.06	4.41	3.69
	80	7.17	3.40	2.89
	100	5.07	2.72	2.50
	120	4.18	2.05	3.17
10	20	6.42	3.51	4.03
	40	6.27	3.62	4.56
	60	8.21	4.62	4.04
	80	8.97	3.66	3.87
	100	6.55	2.40	2.81
	120	5.21	1.18	2.39
12	20	7.89	5.11	4.25
	40	9.16	4.87	4.86
	60	9.52	4.10	4.43
	80	7.81	4.26	4.95
	100	9.21	4.77	4.55
	120	10.19	3.60	3.59

While Meta-RaPS algorithm consistently performed better, its relative improvement over the Partitioning Heuristic declined as the number of jobs increased. On the other hand, it showed more improvement for larger number of machines. This pattern of behavior held true in our computational study irrespective of the dominance criterion. Figure 1 is an example of this trend when the processing times were dominant.

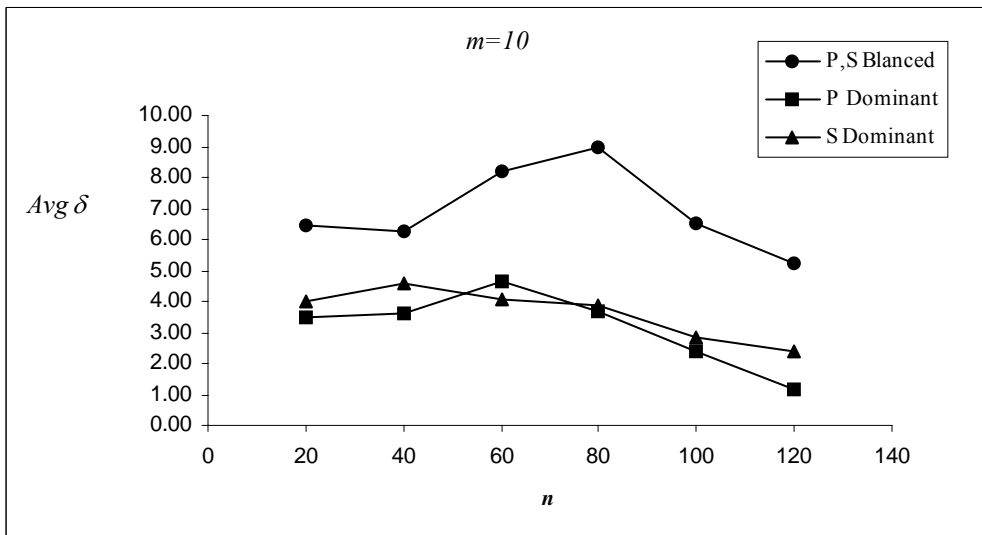


“Fig. 1. Average δ for P_{ij} , Dominant.”

Furthermore, for a fixed number of machines, Meta-RaPS showed more improvement over the Partitioning Heuristic when the processing times and setup times were balanced (see Figures 2). Figures 2 and 3 show examples of this pattern for 4 and 10 machines respectively. The pattern was similar across all machine numbers, and therefore no graphs are presented.



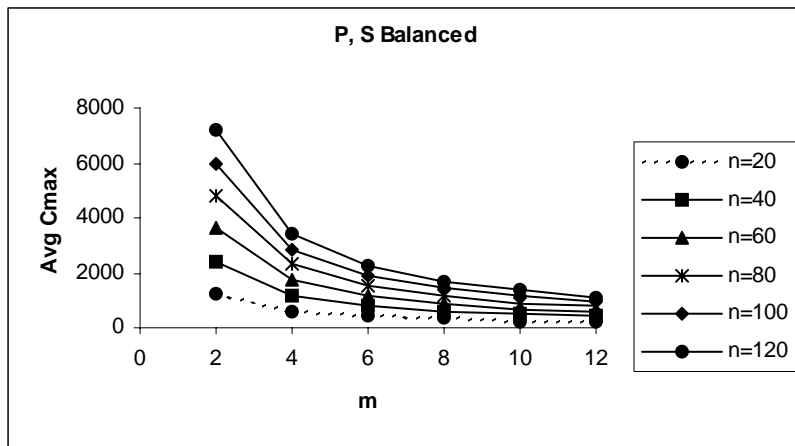
“Fig. 2. The effect of the dominance criterion for 4 machines”



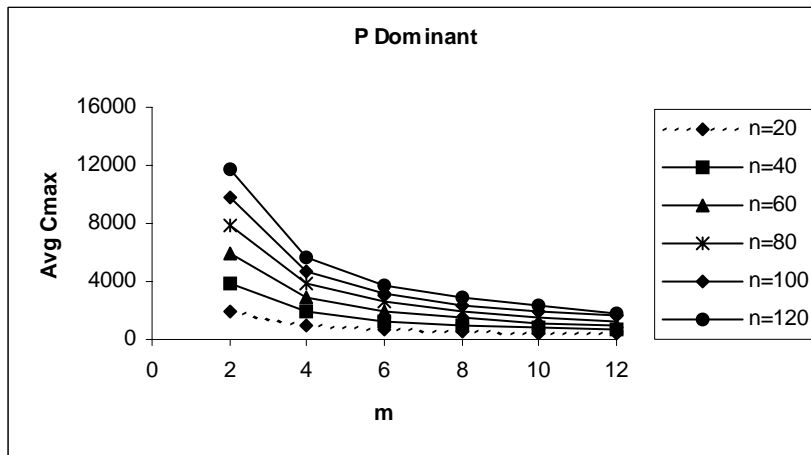
“Fig. 3. The effect of the dominance criterion for 10 machines”

Another measure of performance for the comparison between both heuristics is the number of times one of the heuristics outperformed the other. It turned out that Meta-RaPS outperformed the Partitioning Heuristic in all 2250 instances (small and large) except for the 3 instances.

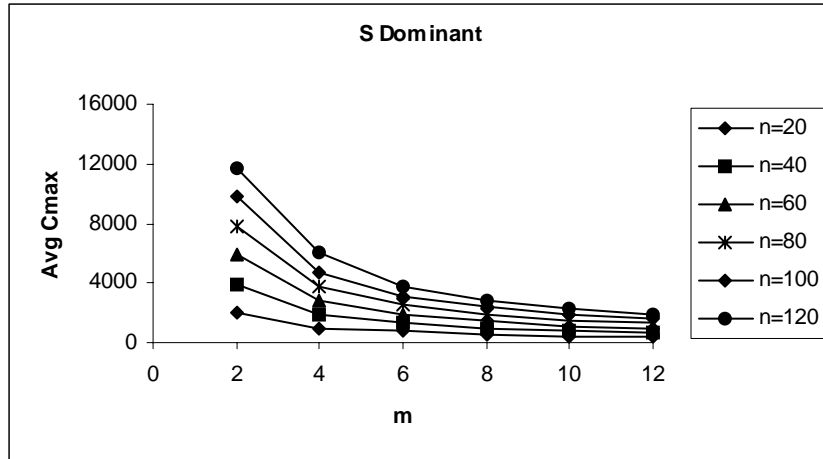
Since it is clear that Meta-RaPS performs better, it will be interesting to analyze it further. The average C_{max} over 15 replications for a certain number of jobs decreased as the number of machines increased (see Figure 4). Also, for every job number, the amount of C_{max} reduction becomes less as the number of machines increases, which is expected; nonetheless, this information helps the decision maker determine the most economical number of machines to own given the amount of reduction in the C_{max} . Figures 4 through 6 depict this relation for the three dominance criteria.



“Fig. 4. Average C_{max} versus m for P_{ij}, S_{ij} Balanced”



“Fig. 5. Average C_{max} versus m for P_{ij} Dominant”



“Fig. 6. Average C_{max} versus m for S_{ij} Dominant”

6. Conclusions and Future work

The problem addressed in this paper is the unrelated parallel machine scheduling problem (PMSP) with machine-dependent and sequence-dependent setup times to minimize the makespan. This problem has been addressed by Al-Salem (2004) where he developed the Partitioning Heuristic to find good solutions for the problem. In that paper, the author compared his solutions with a lower bound (LB) and concluded that although the LB was relatively weak, the Partitioning Heuristic performed well. When the processing times were balanced, the average deviation of the partitioning heuristic from the LB was within 9 to 11 percent, and when the processing times or setup times were dominant the average deviation was within 5 to 7 percent.

Meta-RaPS is a meta-heuristic approach that has been introduced recently and has been developed in this paper for the unrelated PMSP to find solutions with better quality. Integer programming was used to find optimal solutions for small-sized instances of the problem. Meta-RaPS was able to find optimal solutions for all small problems. For large problems, however, Meta-RaPS was compared to the Partitioning Heuristic. For all three

dominance criteria between processing times and setup times, Meta-RaPS outperformed the Partitioning Heuristic in all 2250 instances except for 3 instances.

It will be interesting to extend this work by including the Partitioning Heuristic as a construction heuristic in Meta-RaPS and investigate whether more improvement can be accomplished compared to the current construction heuristic implemented in Meta-RaPS.

Benchmark data sets and solutions for both heuristics used in this paper are made available in Rabadi (2005) for other researchers to compare their solution methodologies with.

References

1. Aiex. R.M., Binato. S. and Resende. M.G.C. (2003) Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 29,393–430.
2. Al-Salem. A. (2004) Scheduling to minimize makespan on unrelated Parallel machines with sequence dependent setup times. *Engineering Journal of the University of Qatar*, 17, 177–187.
3. Allahverdi. A., Gupta J.N.D., Aldowaisan. T. (1999) A review of scheduling research involving setup considerations. *Omega*, 27, 219–39.
4. Arcus. A.L. (1966) COMSOAL: A Computer method of sequencing operations for assembly lines. *International Journal of Production Research*, 4, 259–277.
5. Azizoglu. M. and Kirka. O. (1999) Scheduling jobs on unrelated parallel machines to minimize regular total cost functions. *IIE Transactions*, 31, 153–159.
6. Bank. J. and Werner. F. (2001) Heuristic Algorithms for Unrelated Parallel Machine Scheduling with a Common Due Date, Release Dates, and Linear Earliness and Tardiness Penalties. *Mathematical and Computer Modelling*, 33, 363–383.
7. Bruno. L.G., Coffman. E.G. and Sethi R. (1974) Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17, 382–387.
8. Cheng. T. and Sin. C. (1990) A State-of-the-Art Review of Parallel-Machine Scheduling Research. *European Journal of Operation Research*, 47, 271–292.

9. DePuy. G. and Whitehouse. G. E. (2001) A Simple and Effective Heuristic for the Multiple Resource Allocation Problem. *International Journal of Production Research*, 32, 4, 24–31.
10. DePuy. G.W., Moraga. R.J. and Whitehouse. G.E. (2005) Meta-RaPS: A Simple And Effective Approach For Solving The Traveling Salesman Problem, *Transportation Research Part E: Logistics and Transportation Review*, Vol. 41, No. 2, pp. 115-130.
11. Dhaenens-Flipo. C. (2001) A bicriterion approach to deal with a constrained single-objective problem. *International Journal of Production Economics*, 74, 93-101.
12. Dunstall. S. and Wirth. A. (2005) Heuristic methods for the identical parallel machine flowtime problem with set-up times. *Computers & Operations Research*, 32, 2479–2491.
13. Feo. T. and Resende. M. (1995) Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6,109–133.
14. Feo. T.A. Sarathy. K. and McGahan. J. (1996) A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties. *Computers & Operations Research*, 23, 881–895.
15. Feo. T.A., Venkatraman. K. and Bard. J.F. (1991) A GRASP for a difficult single machine scheduling problem. *Computers & Operations Research*, 18, 635–643.
16. Franca. P.M., Gendreau. M., Laporte G. and Muller F.M. (1996) A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *International Journal of Production Economics*, 43, 79-89.
17. Garey. M.R. and Johnson. D.S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company.
18. Gendreau. M., Laporte. L. and Guimaraes. E.M. (2001) A divide and merge heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 133, 183–189.
19. Ghirardi. M. and Potts. C.N. (2005) Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research*, 165, 2, 457–467.
20. Glass. C.A., Potts. C.N. and Shade. P. (1994) Unrelated Parallel Machine Scheduling Using Local Search. *Mathematical and Computer Modeling*, 20, 2, 41–52.
21. Graves. S.C. (1981). A review of Production Scheduling. *Operation Research*, 29, 646 – 675.

22. Guinet. A. (1991) Textile Production Systems: a succession of Non-identical Parallel Processor Shops. *Journal of Operational Research Society*, 42(8), 655–671.
23. Hariri. A.M.A. and Potts. C.N. (1991) Heuristics for scheduling unrelated parallel machines. *Computers and Operations Research*, 18, 3, 323–331.
24. Horn. W.A. (1973) Minimizing average flow time with parallel machines. *Operations Research*, 21, 846–7.
25. Karp. R.M. (1972) Reducibility among combinatorial problems, in R.E. Miller and J.W. Thatcher. Eds.: *Complexity of Computer Computations*, Plenum Press, New York, 85 – 103
26. Kim. D.W., Kim. K.H., Jang. W. and Chen. F.F. (2002) Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer Integrated Manufacturing*, 18, 3-4, 223–231.
27. Kim. D.W., Na. D.G. and Chen. F.F. (2003) Unrelated parallel machine scheduling with setup times and total weighted tardiness objective. *Robotics and Computer Integrated Manufacturing*, 19, 1-2, 173–181.
28. Kurz. M.E. and Askin. R. G. (2001) Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research*, 39, 3747-3769.
29. Laguna. M. and González-Velarde. J.L. (1991) A search heuristic for just-in-time scheduling in parallel machines. *Journal of Intelligent Manufacturing*, 2, 253–260.
30. Lancia. G. (2000) Scheduling jobs with release dates and tails on two unrelated parallel machines to minimize the makespan. *European Journal of Operational Research*, 120, 277–288.
31. Lawler. E.L., Lenstra. J.K., Rinnooy Kan. A.H.G. and Shmoys. D.B. (1993) Sequencing and Scheduling: Algorithms and complexity. *Handbooks in Operations Research and Management Science 4, Logistics of Production and Inventory*, North Holland, Amsterdam, 445–524.
32. Lee. H., Bhaskaran. K. and Pinedo. M. (1997) A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions*, 100, 464–474.
33. Liaw. C.F., Lin. Y.K., Chen. C.Y. and Chen. M. (2003) Scheduling unrelated parallel machines to minimize total weighted tardiness. *Computers & Operations Research*, 30, 1777–1789.
34. Lin. Y. and Wenhua. L. (2004). Parallel machine scheduling of machine-dependent jobs with unit-length. *European Journal of Operational Research*, 156, 261–266.

35. Martello. S., Soumis. F. and Toth P. (1997) Exact and approximation algorithms for makespan minimization on unrelated parallel machines. *Discrete Applied Mathematics*, 75 (1997) 169-188.
36. McNaughton. R. (1959) Scheduling with deadlines and loss function. *Management Science*, 6, 1–12.
37. Mokotoff. E. (2001) Parallel Machine Scheduling Problems: A Survey, Asia-Pacific. *Journal of Operational research*, 18, 193–242.
38. Moraga. R.J., DePuy. G.W. and Whitehouse. G.E. (2005) Meta-RaPS Approach for the 0-1 Multidimensional Knapsack Problem, *Computers and Industrial Engineering*, Vol. 48, No. 2, pp 83-96.
39. Moraga. R.J. (2002) *Meta-RaPS: An Effective Solution Approach for Combinatorial Problems*, Ph.D. thesis. Orlando, FL: University of Central Florida.
40. Pinedo. M. (2002) *Scheduling: Theory, Algorithms, and Systems*, 2nd edition, Printce Hall, New Jersey.
41. Rabadi. G. Mollaghasemi. M. and Anagnostopoulos. G.C. (2004) A Branch-and-Bound Algorithm for the Early/Tardy Machine Scheduling Problem with a Common Due-Date and Sequence-Dependent Setup Time. *Computers & Operations Research Journal*, 31, 10, 1727 – 1751.
42. Rabadi G. (2005) <http://www.SchedulingResearch.com>, a web site that includes benchmark problem data sets and solutions for scheduling problems.
43. Radhakrishnan. S. and Ventura. J.A. (2000) Simulated annealing for parallel machine scheduling with earliness/tardiness penalties and sequence-dependent set-up times. *International Journal of Production Research*, 38, 2233-2252.
44. Rojanasoonthon. S. and Bard. J.F. (2005) A GRASP for parallel machine scheduling with time windows. To appear in, v17, n 1, *INFORMS Journal on Computing*.
45. Randhawa. S. U. and Kuo C. H. (1997) Evaluating scheduling heuristics for non-identical parallel processors. *International Journal of Production Research*, 35, 969-981.
46. Srivastava. B. (1997). An effective heuristic for minimizing makespan on unrelated parallel machines. *Journal of the Operational Research Society*, 49, 886–894.
47. Van Hentenryck. P. (2001) *ILOG OPL Studio 3.5 Language Manual*. ILOG, France.

48. Weng. M., Lu. J. and Ren. H. (2001) Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. *International Journal of Production Economics*, 70, 215–226.