# A Logic-Based Benders Approach to Scheduling with Alternative Resources and Setup Times

**Tony T. Tran** and **J. Christopher Beck**
Department of Mechanical and Industrial Engineering
University of Toronto, Toronto, Ontario, Canada
{tran,jcb}@mie.utoronto.ca

**Abstract**

We study an unrelated parallel machines scheduling problem with sequence and machine dependent setup times. A logic-based Benders decomposition approach is proposed to minimize the makespan. The decomposition approach is a hybrid model that makes use of a mixed integer programming master problem and a travelling salesman problem subproblem. The master problem is a relaxation of the problem and is used to create assignments of jobs to machines, while the subproblem obtains optimal schedules based on the master problem assignments. Computational results comparing the Benders decomposition and mixed integer program formulation show that the Benders model is able to find optimal solutions to problems up to five orders of magnitude faster as well as solving problems four times the size possible previously.

## 1 Introduction

In many practical scheduling problems, the scheduler is faced with both resource alternatives and sequence dependent setup times. That is, a job may be assigned to one of a set of resources and consecutive jobs on the same resource must have a minimum setup time between them. For example, in a chemical plant, reactors must be cleaned when changing from processing one mixture to another. The cleaning times may depend on which specific job comes before the cleaning and which comes after. If the preceding chemical affects the succeeding one, cleaning may take longer to ensure that the reactor is properly prepared. The products processed in the reverse order may not have the same ill effects because the offending product in the previous example may not suffer the same contamination and so, cleaning may take less time. Further examples can be found in the plastic, glass, paper and textile industries where setup times of significant length exist (França et al. 1996). Allahverdi et al.(1999) review the importance of setup times in real world problems.

This paper addresses the unrelated parallel machines scheduling problem (PMSP) with machine and sequence dependent setup times. In this problem, jobs must be assigned to one of a set of alternative resources. Jobs assigned to the same resource have a setup time which is defined as the time that must elapse between the end of one job and the start of the next. This setup time is sequence and machine dependent in that the elapsed time between jobs $j$ and $k$ will differ depending on whether $j$ precedes $k$ or $k$ precedes $j$ and which machine the pair of jobs are assigned to. We concern ourselves with minimizing the makespan or the maximum completion time of a schedule, $C_{max}$, as the objective function. Using the three-field notation given by Graham et al. (1979), this problem can be denoted as (R|sds|C$max$).

We develop an exact method to solve the PMSP based on logic-based Benders decomposition. This approach was developed to verify logic circuits by Hooker (1995) and further explained later (Hooker and Ottosson 2003). Hooker (2005) applied logic-based Benders decomposition to a problem similar to the PMSP with release and due dates, but without setup times. The Benders decomposition introduced in this paper makes use of a Mixed Integer Program (MIP) master problem and either a Constraint Program (CP) solver or a specialized Travelling Salesman Problem (TSP) solver for the subproblems. The MIP model is used to find tight lower bounds and machine assignments, while the CP or TSP solvers sequence the jobs on machines. The new model is compared to an existing MIP model which finds the optimal solution.

## 2 Background

In this section, we will define the PMSP with sequence and machine dependent setup times. A review of related work is presented and finally, an existing MIP model to solve the PMSP is shown.

### 2.1 Problem Definition

In the PMSP, a set of $N$ jobs are to be scheduled on $M$ machines with the objective of minimizing the makespan. Each job has a processing time $p_{ij}$, the time to process job $j$ on machine $i$. The machines in this system are unrelated, i.e., a job $j$ can have a processing time greater than $k$ on one machine, but the reverse could be true on another machine. There is a sequence and machine dependent setup time, $s_{ijk}$, which is the time that must elapse before a machine can begin processing job $k$ if job $j$ precedes it on machine $i$. The setup times are assumed to follow the triangle inequality $s_{ijk} \leq s_{ilk} + s_{ljk}$ and are only incurred when switching from one job to another. The goal of the problem is to determine how to assign jobs to machines and then sequence these jobs, independently on each machine.

## 2.2 Related Work

Most research has been focused on the PMSP with identical machines (Graves 1981; Cheng and Sin 1990; Dunstall and Wirth 2005; Kurz and Askin 2001). Research on the PMSP with unrelated machines has concentrated on the problem without setup times. An exact algorithm was developed by Lancia (2000) to minimize makespan. A genetic algorithm, simulated annealing, and tabu search were compared by Glass et al. (1994).

The PMSP with sequence and machine dependent setup times is strongly NP-Hard because the single machine scheduling problem with sequence dependent setup times (1|sds|C*max*) is equivalent to a travelling salesman problem (TSP) (Baker 1974). Thus, the PMSP with setup times can be thought of as an allocation and routing problem where cities are allocated to salesmen who then must find their own tour. Even in the case where all machines are identical (P|sds|C*max*), the problem is strongly NP-hard (França et al. 1996; Mendes et al. 2002). The combinatorial complexity of the PMSP has resulted in little research in exact optimization methods. A MIP model does exist (Guinet 1991) which obtains the optimal schedule for the PMSP with setup times with total completion time or total tardiness as objectives. However, this MIP model is only able to solve for small instances:9 jobs, 2 machines or 8 jobs, 4 machines. The sizes of these problems are not very practical for real life use where the number of jobs can be much larger.

Al-Salem (2004) developed a constructive heuristic named the partitioning heuristic to solve large instances of the PMSP with setup times. Helal et al. (2006) developed a tabu search to solve the same problem. Rabadi et al.(2006) presented a meta-heuristic titled Meta-RaPS to minimize makespan. An ant colony optimization method was implemented and shown to perform better then the partitioning heuristic, tabu search, and Meta-RaPS for the unrelated PMSP with setup times (Arnaout, Rabadi, and Musa 2008). In all these studies, optimal makespans were not the goal because the problem sizes were too large for current methods to find optimality. The performance of the heuristics was evaluated by comparison of the solutions they provided with lower bounds on the makespan for any instances with more than 10 jobs. All heuristics tested performance for instances of sizes up to 100 jobs and 10 machines; Rabadi et al. (2006) and Arnaout et al. (2008) tested problems of 120 jobs and 12 or 8 machines respectively.

Focacci et al. (2000) proposed a two phase algorithm based on CP to optimize both the makespan and the sum of setup times for a similar problem to the PMSP with sequence dependent setup times. In this paper, jobs consist of multiple activities and precedence constraints exist between these activities. In the first phase of the algorithm, a time limited, incomplete branch-and-bound method is used to find solutions with small makespan. The second phase minimizes the sum of setup times with the constraint that any schedule found must have a makespan equal to or less than the makespan found in the first phase. They run their model on problems of up to 16 jobs, each consisting of 12 activities, and 16 machines.

## 2.3 Mixed Integer Programming Model

A MIP model used to find optimal solutions for the unrelated PMSP with setup times is presented by various researchers (Helal, Rabadi, and Al-Salem 2006; Rabadi, Moraga, and Al-Salem 2006). This formulation is based on a similar problem by Guinet (1991) with different objectives of total completion time or total tardiness.

$$\min \quad C_{max}$$

$$\text{s.t.} \quad \sum_{j=0, j\neq k}^{N} \sum_{i=1}^{M} x_{ijk} = 1, \qquad k \in N \qquad (1)$$

$$\sum_{j=0, j\neq h}^{N} x_{ijh} = \sum_{k=0, k\neq h}^{N} x_{ihk}, \qquad h \in N, i \in M \qquad (2)$$

$$C_k \geq C_j + \sum_{i=1}^{M} x_{ijk}(s_{ijk} + p_{ik}) + V(\sum_{i=1}^{M} x_{ijk} - 1)$$
$$j \in N, k \in N \qquad (3)$$

$$\sum_{j=0}^{N} x_{i0j} = 1, \qquad i \in M \qquad (4)$$

$$C_j \leq C_{max}, \qquad j \in N \qquad (5)$$

$$C_0 = 0 \qquad (6)$$

$$C_j \geq 0, \qquad j \in N \qquad (7)$$

$$x_{ijk} \in (0;1), \qquad j \in N, \; k \in N, i \in M \qquad (8)$$

where

| | |
|---|---|
| $C_{max}$: | Maximum completion time (makespan) |
| $C_j$: | Completion time of job $j$ |
| $x_{ijk}$: | 1 if job $k$ is processed directly after job $j$ on machine $i$ |
| $x_{i0k}$: | 1 if job $k$ is the first job to be processed on machine $i$ |
| $x_{ij0}$: | 1 if job $j$ is the last job to be processed on machine $i$ |
| $s_{i0k}$: | setup time before job $k$ if it is the first job on machine $i$ |
| V: | A large positive number |

Constraint (1) ensures that each job is scheduled on a single machine only and after exactly one other job. Constraint (2) ensures that each job cannot be preceded or succeeded by more than one job. Constraint (3) sets the completion times of each job such that if job $j$ precedes job $k$, job $k$ cannot also precede job $j$ to create an infeasible cycle. If job $k$ is processed directly after job $j$, $\sum_{i=1}^{M} x_{ijk} - 1 = 0$ and the constraint makes it so that $C_k \geq C_j + s_{ijk} + p_{ik}$ with $i$ being the machine on which the two jobs are assigned. If job $k$ is not scheduled directly after job $j$ on a machine, $\sum_{i=1}^{M} x_{ijk} - 1 = -1$ and the large $V$ term makes the constraint redundant. Constraint (4) guarantees that only one job can be scheduled first on each machine. Constraint (5) sets the makespan to be at least as large as the largest completion time of all jobs. Constraint (6) sets the completion time of job 0, an auxiliary job used to enforce the start of a schedule, to zero and constraint (7) ensures positive completion times. Constraint (8) defines the decision variables as binary.

## 3 Logic-Based Benders Decomposition

The PMSP with setup times can be decomposed into an assignment master problem and sequencing subproblem. In the assignment master problem, the jobs are assigned to machines. This assignment results in multiple subproblems where each machine is a scheduling problem to sequence the assigned jobs. A MIP model is presented for the master problem. The use of MIP takes advantage of operations research tools to obtain tight lower bounds on the PMSP with setup times. Sequencing is accomplished in the subproblem where CP and TSP solvers are more adept to obtaining answers quickly.

### 3.1 Assignment Master Problem

The MIP formulation of the master problem is a relaxation of the PMSP with setup times. In this relaxation, jobs are assigned to machines, but instead of solving for a single sequence of jobs on each machine, many smaller subsequences are allowed. The setup times are calculated for each subsequence; their sum is a lower bound on the actual total setup time on a machine. This assignment and subsequencing leads to an infeasible schedule if multiple subsequences are created. However, the relaxation gives a tighter lower bound than if setup times are completely ignored, while being significantly less difficult than the full problem. The master problem is,

$$\min \quad C_{max}$$

$$\text{s.t.} \quad \sum_{j \in N} x_{ij} p_{ij} + \xi_i \leq C_{max} \qquad i \in M \qquad (9)$$

$$\sum_{i \in M} x_{ij} = 1 \qquad j \in N \qquad (10)$$

$$\xi_i = \sum_{j \in N} \sum_{k \in N, k \neq j} y_{ijk} s_{ijk} \qquad i \in M \qquad (11)$$

$$x_{ik} = \sum_{j \in N} y_{ijk} \qquad k \in N; i \in M \qquad (12)$$

$$x_{ij} = \sum_{k \in n} y_{ijk} \qquad j \in N; i \in M \qquad (13)$$

$$cuts \qquad (14)$$

$$x_{ij} \in \{0;1\} \qquad j \in N; i \in M \qquad (15)$$

$$0 \leq y_{ijk} \leq 1 \qquad j,k \in N; i \in M \qquad (16)$$

where

| | |
|---|---|
| $C_{max}$: | Makespan of the master problem |
| $\xi_i$: | Total setup time incurred from all sequences on machine $i$ |
| $x_{ij}$: | 1 if job $j$ is processed on machine $i$ |
| $y_{ijk}$: | 1 if job $k$ is processed directly after job $j$ on machine $i$ |
| $y_{i0k}$: | 1 if job $k$ is processed first on machine $i$ |
| $y_{ij0}$: | 1 if job $j$ is process last on machine $i$ |

The makespan on each machine with the relaxed setup times is defined in constraint (9) as the summation of processing times for all jobs that are assigned to that machine and the relaxed total setup times. Setting the makespan to be greater than or equal to the relaxed makespan of each machine enforces that the MIP model optimizes the maximum makespan across all machines. Constraint (10) ensures that each job is assigned to exactly one machine. Constraint (11) assigns the relaxed setup time of a machine $i$, $\xi_i$, to be a lower bound on the additional time required from the sequencing of jobs, $y_{ijk}$, and their respective setup times, $s_{ijk}$. The relaxation of setup times allows, instead of a sequence of jobs from the first to last job processed on a machine, many smaller sequences independent of each other. For example, given jobs $j$, $k$, $j_3$, $j_4$, and $j_5$, a feasible sequence is [*start* - $j$ - $k$ - $j_3$ - $j_4$ - $j_5$ - *end*]. However, (12) and (13) set each job to have exactly one other job scheduled directly before and after it without the restriction of cycles as was seen in constraint (3). This will make it possible to assign two sequences, [*start* - $j$ - $k$ - $j_3$ - *end*] and the cycle [$j_4$ - $j_5$ - $j_4$ - $j_5$...]. Constraint (14) are *cuts* added to the master problem from the subproblem each time an infeasible solution is found. In the first iteration of the master problem, the set of *cuts* is empty. The last constraints, (15) and (16), force the decision variables $x_{ij}$ to be binary, i.e., either a job $j$ is assigned machine $i$ or not and $y_{ijk}$ to be between 0 and 1.

This formulation is equivalent to solving the (R|sds|C*max*), but instead of solving for the exact single sequence of jobs to process on a machine, many subsequences are allowed which will include all jobs. This relaxation creates a tight lower bound for the actual makespan of a machine and is similar to solving the assignment problem. Therefore, the makespan found from solving the master problem may be infeasible given a proper sequencing of jobs.

### 3.2 Sequencing Subproblem

Once a solution of the master problem is found, the set of jobs to schedule on each machine is known. These sets of jobs create $m$ subproblems, one for each machine. In this section, two different subproblem formulations are presented: a CP and a TSP model. Both models will create a sequence of jobs on a machine such that the makespan is minimized. This objective can also be thought of as minimizing the sum of setup times since the set of jobs to be processed and their processing times are already determined.

**Constraint Program** Let $t_j$ and $e_j$ be the start and end times for job $j$ respectively. $C^{hi}_{max}$ represents the makespan for machine $i$ in iteration $h$. The CP formulation of the subproblem is shown below.

$$\min \quad C^{hi}_{max}$$

$$\text{s.t.} \quad t_j + p_{ij} = e_j \qquad j \in N' \qquad (17)$$

$$t_j \geq e_k + s_{ijk} \vee t_k \geq e_j + s_{jk} \qquad k, j \in N'; k \neq j \qquad (18)$$

$$t_0 = 0 \qquad (19)$$

$$\text{disjunctive}(t_j, p_{ij}) \qquad (20)$$

The objective of the subproblem is to sequence the jobs in such a way as to minimize the makespan. The set of jobs to schedule is $N'$ which are the jobs chosen in the master problem with an additional auxiliary job (job 0) which has setup times and processing times equal to 0. This extra job acts as the first job in the sequence which incurs no setup times for the job that is scheduled after it. Constraint (17) sets the end time of each job to be the start time plus the processing time. Constraint (18) ensures that setup times are adhered to. If a job $k$ is processed directly after a job $j$, then constraint (18) becomes an equality constraint. In the case where job $k$ is not directly processed after a job $j$, but still processed later, the constraint holds given that setup times adhere to the triangle inequality. Constraint (19) sets the start time of job 0 to zero. Finally, constraint (20) is a global constraint that ensures the unary resource constraint is satisfied.

We implement this model in IBM ILOG Scheduler. Each job is an activity in IBM ILOG Scheduler, represented by the variables $t_j$ and $e_j$. The machine that these jobs are assigned to is represented by a unary resource and the setup times, $s_{jk}$, are assigned as the elements of an IloTransitionParam matrix. The IloTransitionParam matrix is an asymmetric square matrix that defines the sequence dependent setup times in IBM ILOG Scheduler for each activity pair.

**Travelling Salesman Problem** We know that the sequencing of jobs on a single machine is equivalent to a TSP with directed edges, also known as an asymmetric TSP. In this TSP, jobs are the nodes and distances between nodes are the setup time between the two connected jobs and the processing time of the job which is the start node. This representation ensures that travelling along any edge from node $a$ to node $b$ will contribute the cost of processing job $a$ and the setup time from job $a$ to job $b$. With this translation, it is possible to see that the setup time problem on a single machine is equivalent to a TSP and a cycle of a TSP from a start node to all other nodes and back to the initial start node is the sequence of jobs to process and the distance travelled being the makespan. This representation is shown in Figure 1.

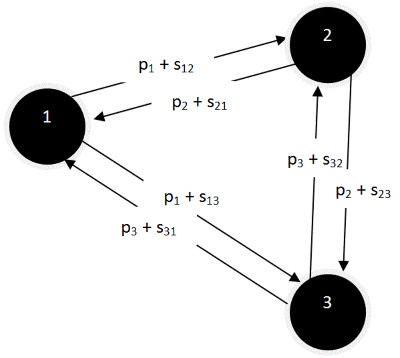

Figure 1: TSP representation

It can be seen that if the order of jobs to be processed is 1, 3 then 2, the distance travelled would be, $p_1 + s_{13} + p_3 + s_{32} + p_2$. This tour distance is equal to the makespan of processing jobs in that order. Therefore, it may be better to use a TSP solver which has been optimized to solve such problems in place of a generic CP solver which is more expressive, but not as good at solving this problem.

**Feasible Schedules from the Subproblem** Solving the sequencing subproblem leads to a feasible schedule. Often with Benders decomposition, a model is searching in the infeasible region of solutions and the first feasible solution found is the optimal one. In our logic-based Benders decomposition approach, while the search is performed in the infeasible region, the sequencing subproblem solves the local schedule once the jobs are allocated to machines. The solution from the subproblem creates a feasible schedule with a makespan equal to the largest makespan found across all subproblems. Therefore, it is possible to stop the solving at any time after the first complete iteration and obtain a feasible schedule. This schedule may not be optimal if the problem solving is stopped prematurely. However, it is possible to compare this value against the makespan found from the most recent master problem solution to calculate an upper bound on how far the current schedule is from optimality. Therefore, the Benders decomposition will store the best solution found so far. At the completion of all subproblems during an iteration, the schedule created will be compared to the best solution found so far and it will be updated if necessary.

## 3.3 Cuts

If the makespan found in the subproblem is less than or equal to the master problem's makespan $C_{max}$, then this subproblem is feasible and no cuts are added to the master problem. In the case where the makespan found is greater than $C_{max}$, a cut is created and sent to the master problem. The master problem is then re-solved with the added cut. The cut from such a subproblem in an iteration $h$ is,

$$C_{max} \geq C^{hi*}_{max}[1 - \sum_{j \in N'} (1 - x_{ij})] \quad i \in M \quad (C1)$$

Here, $C_{max}$ is the makespan variable in the master problem and $C_{max}^{hi*}$ is the makespan found in iteration $h$ when solving the subproblem for machine $i$. The cut states that the future solutions of the master problem can only decrease the makespan if another assignment of jobs is given. That is, if the same assignment is given to the subproblem, the $x_{ij}$ variables that are part of this cut will all equal to 1. If this is the case, then $(1 - x_{ij}) = 0$ for all $j$ and the makespan of the subproblem becomes a lower bound on $C_{max}$. When a different assignment is made and at least one of the $x_{ij}$ variables that previously had a value of 1 is 0, the cut becomes redundant as the right hand side will be at most zero. This cut follows the 2 conditions defined by Chu and Xia (2005) to be a valid cut; the cut removes the current solution from the master problem and does not eliminate any global optimal solutions.

The cut presented is a type of *nogood* cut (Hooker 2005), stating that the current solution is infeasible and so is removed from the search space. We can tighten the cut by introducing the values $\mathit{minPre}_{hj}$, $\mathit{minSuc}_{hj}$, $\mathit{maxPre}_{hj}$, $\mathit{maxSuc}_{hj}$ and $\mathit{minTran}_{hj}$. $\mathit{minPre}_{hj}$ and $\mathit{maxPre}_{hj}$ are the minimum and maximum setup times if job $j$ directly succeeds another job that is assigned to the same machine in iteration $h$ respectively. Similarly, $\mathit{minSuc}_{hj}$ and $\mathit{maxSuc}_{hj}$ are the minimum and maximum setup times if job $j$ directly precedes another job that is assigned to the same machine in iteration $h$. Finally, $\mathit{minTran}_{hj}$ is the minimum setup time between any two jobs in $N'$ excluding job $j$. These values are defined as,

$$\begin{aligned}
\mathit{minPre}_{hj} &= MIN_{k\in N';k\neq j}(s_{ikj})\\
\mathit{minSuc}_{hj} &= MIN_{k\in N';k\neq j}(s_{ijk})\\
\mathit{maxPre}_{hj} &= MAX_{k\in N';k\neq j}(s_{ikj})\\
\mathit{maxSuc}_{hj} &= MAX_{k\in N';k\neq j}(s_{ikj})\\
\mathit{minTran}_{hj} &= MIN_{k\in N';l\in N';k\neq l}(s_{ikl})
\end{aligned}$$

A cut that uses more information is to find a tight upper bound on the total reduction of the makespan when a job is removed from a schedule. To find this upper bound, we work backwards and calculate the increase in the makespan if job $j$ is added to an optimal sequence consisting of the jobs in $N'$ except $j$. Assume that the optimal makespan of the schedule, $C'$, is known. If job $j$ is inserted into the schedule greedily, the insertion may occur at either a position that minimizes the setup time to job $j$ or from job $j$. In the worst case scenario, if the insertion that occurred to minimize the setup to (from) job $j$, the job that succeeds (precedes) $j$ may result in the largest setup time from (to) job $j$. This insertion will further remove a single setup time since job $j$ may be scheduled between any two other jobs. In the worst case, the removed setup time is the minimum setup time between any two jobs in $N'$. We know that this insertion results in a feasible schedule and in the best case will give the optimal makespan for scheduling $N'$. The increase in makespan of adding job $j$ is denoted as $\delta_{hij}$. We know that,

$$C_{max}^{hi*} \leq C' + \delta_{hij}$$

which rearranged is,

$$C' \geq C_{max}^{hi*} - \delta_{hij}$$

Therefore, if the optimal makespan of a schedule consisting of all jobs in $N'$ is known, removing $\delta_{hij}$ from the optimal makespan is guaranteed to result in a makespan less than or equal optimal if job $j$ is removed. The cut can then be,

$$C_{max} \geq C_{max}^{hi*} - \sum_{j\in N'}(1 - x_{ij})\delta_{hij} \quad i \in M \quad (C2)$$

where,

$$\begin{aligned}
\delta_{hij} = \quad & p_{ij} - \mathit{minTran}_{hj}\\
& + MIN[(\mathit{minPre}_{hj} + \mathit{maxSuc}_{hj}),\\
& \qquad (\mathit{maxPre}_{hj} + \mathit{minSuc}_{hj})]
\end{aligned}$$

A third cut is developed to further improve upon (C2). This cut attempts to not only include extra information about the jobs being assigned, but also account for the jobs that are not assigned to the subproblem machine. This is done by incrementing the makespan by the processing time of a job if that job is to be assigned to a machine. The cut is then,

$$\begin{aligned}
C_{max} \geq C_{max}^{hi*} &+ \textstyle\sum_{k\notin N'} x_{ik}p_{ik}\\
&- \textstyle\sum_{j\in N'}(1 - x_{ij})\delta_{hij} \qquad i \in M \quad (C3)
\end{aligned}$$

## 3.4 Stopping Condition

The Benders approach will iterate between master problem and subproblems until an optimal solution is found and proved. Optimality is proven if one of two conditions is met. The first condition that can prove optimality is if all subproblems solved during an iteration find makespans less than or equal to the $C_{max}$ from the master problem. This solution is optimal because the master problem provides a lower bound on the achievable makespan of the problem. If all subproblems prove that a schedule can be created with the makespan less than or equal to $C_{max}$ of the master problem, it is proven that the schedule is optimal. The second condition that can prove optimality and provide a stopping condition requires that the $C_{max}$ found from the master problem be equal to the best feasible makespan found so far as defined in Section 3.2.

## 4 Computational Results

The Benders decomposition model and MIP model were tested on an Intel Pentium 4 CPU 3.00GHz Hypterthread Tech with 2 MB cache per core, 1 GB of main memory, running Red Hat 3.4.6-3. The MIP master problem and MIP model were implemented with IBM ILOG CPLEX 12.1 and the CP subproblem was implemented with IBM ILOG Solver 6.7 and IBM ILOG Scheduler 6.7. The TSP solver used was tsp_solve.[1] Experiments were run for problem instances of 10, 20, 30, and 40 jobs. For each job size, between 2 and 5 machines were tested. Each of these combinations

[1]A free TSP solver available online at (http://www.or.deis.unibo.it/research_pages/tspsoft.html) which is written in C++.

had a total of 10 instances for a total of 160 instances. A time limit of 3 hours was used. Processing times for each machine job pair were generated from a uniform distribution between 1 and 100. To obtain setup times that were sequence dependent and follow the triangular inequality assumption, each job was given two different sets of coordinates on a Cartesian plane for every machine. The setup times are the Manhattan distances from one job's coordinates to the other's. Distances between the second set of coordinates is used to provide asymmetric setup times. For example, job 0 and job 1 would be given coordinates $\chi_{0a}$, $\chi_{0b}$, $\psi_{0a}$, $\psi_{0b}$, $\chi_{1a}$, $\chi_{1b}$, $\psi_{1a}$, and $\psi_{1b}$. Setup time from job 0 (1) to job 1 (0) would then be $|\chi_{0a} - \chi_{1a}| + |\psi_{0a} - \psi_{1a}|$ ($|\chi_{0b} - \chi_{1b}| + |\psi_{0b} - \psi_{1b}|$).

Table 1 shows results comparing the MIP model, CP Benders, and TSP Benders. In both Benders decomposition models, cut (C2) was used. For these results, the time until an optimal solution was found and proved were recorded. Where the solving timed out, 3 hours was used.

The Benders decomposition model's results, both CP and TSP versions, are a significant improvement over the MIP performance. It is clear that the Benders decomposition approach is capable of solving much larger problems in significantly shorter run times. The Benders decomposition approach, with a CP solver, solves up to 20 jobs in the time limit and the majority of instances of up to 30 jobs. Replacing the CP solver with a TSP solver, the Benders model is able to increase the number of solvable jobs up to 30 consistently. This is in contrast to the MIP model which is able to solve only 10 jobs in the 3 hour time limit.

We see that increasing the number of machines has a greater effect on the performance of the models than increasing the number of jobs. In both Benders models, the master problem had difficulty solving for increased machines. In fact, the TSP subproblem is able to solve each subproblem in milliseconds while the MIP master problem can spend hours searching for an assignment. The opposite is seen for the MIP model. The MIP model has difficulties sequencing large number of jobs on machines and so, in the case where there are only 2 machines, the MIP model has a very high runtime for instances of 10 jobs and 2 machines. When the number of machines is increased to 5, the sequencing problem is simpler and results in fast runtimes.

Using the Benders decomposition model with the TSP subproblem, the three different cuts presented in Section 3.3 are tested on the same set of instances for problem sizes of 10, 20, and 30 jobs. The results are presented in table 2.

Table 2 shows that the performance of cut (C2) is the best overall. In most cases, the difference is not significant, but there are cases where clear differences are found. Specifically, (C2) is able to, in a small number of instances, create a cut that removes an assignment that (C1) would not. This improved cut reduces the total number of iterations required and the time needed to solve the problem. In the 20 jobs and 5 machines test case, the average number of iterations for (C1) is 4.4 while cut (C2) was able to decrease this value to 4. This resulted in a 30 second runtime difference on average to reduce the runtime by about 25%.

One would then assume that cut (C3), using more information, would create better cuts. However, the results showed that cut (C3) performs worst than cut (C2). This is because cut (C3) created a more difficult master problem to solve increasing the total solve time per iteration, while not becoming much tighter than (C2) to reduce the overall number of iterations. The test case with 20 jobs and 5 machines shows how (C3) only reduces the number of iterations by a very small amount, a further 0.2 iterations from cut (C2), but increases on average by 2 seconds of runtime. Including into the cut all jobs that were not assigned to a machine accounts for more information, but proves detrimental to the performance of the Benders model.

In the 30 jobs and 4 machines case, we even see cut (C3) increasing the number of iterations over cut (C2). This seems to contradict the fact that (C3) is a tighter cut. The increase in iteration occurs because some degeneracy exists in the problem. Varying the cut may lead to different assignments in the Benders master problem with equal objective functions. In rare instances, it is possible that the other cuts will lead to the optimal assignment while (C3) leads to an equivalent master problem that does not extend to a global solution.

## 5 Future Work

Though the Benders decomposition approach obtains significant speed ups, the problem sizes that can be solved are limited compared to what heuristic models are currently solving. For larger problems, the Benders decomposition is not able to complete a single instance of the master problem. Problem instances of 100 jobs are tested in previous papers using heuristics and local search (Helal, Rabadi, and Al-Salem 2006; Rabadi, Moraga, and Al-Salem 2006). Specifically, in work done by Helal et al. (2006), schedules for problem instances of 100 jobs and 10 machines are found within minutes. The quality of these schedules is difficult to assess, given that the optimal solutions are not known. However, for smaller instances (8 jobs and 4 machines) the optimal solution was known from the MIP model and the heuristic used was experimentally shown to have on average 2.5% deviation from optimal.

If optimal solutions are not possible because of large instances, it is clear that heuristic solutions are necessary. Stopping the Benders program early and using the best found schedule as shown in Section 3.2 is one approach to increase the size of problems for which the Benders model can obtain schedules. However, this approach is only useful if the problems are small enough such that the Benders model can solve for one complete iteration in a reasonable time. From our experiments, we found some instances of 40 jobs and 5 machines where solving for one complete iteration took more than one hour. Solve times of the subproblem are almost instantaneous when the TSP solver is used on all instances, but the MIP master problem may be intractable once the size of the problem reaches 50 or more jobs and 5 machines. This means that for problems as large as 100 jobs, the Benders decomposition model is not likely to solve the master problem within the time limit.

Therefore, we plan to investigate not solving the master problem all the way to optimality. This would reduce

|  |  | MIP |  | CP Benders |  | TSP Benders |  |
|---|---|---|---|---|---|---|---|
| n | m | Avg Runtime | # uns. | Avg Runtime | # uns. | Avg Runtime | # uns. |
| 10 | 2 | 9885.14 | 9 | 0.24 | 0 | 0.12 | 0 |
|  | 3 | 7924.16 | 7 | 0.29 | 0 | 0.22 | 0 |
|  | 4 | 1169.69 | 1 | 0.55 | 0 | 0.42 | 0 |
|  | 5 | 13.59 | 0 | 0.56 | 0 | 0.45 | 0 |
| 20 | 2 | 10800.00 | 10 | 9.79 | 0 | 1.08 | 0 |
|  | 3 | 10800.00 | 10 | 6.41 | 0 | 2.02 | 0 |
|  | 4 | 10800.00 | 10 | 239.15 | 0 | 53.12 | 0 |
|  | 5 | 10800.00 | 10 | 509.85 | 0 | 122.46 | 0 |
| 30 | 2 | 10800.00 | 10 | 7997.13 | 5 | 2.15 | 0 |
|  | 3 | 10800.00 | 10 | 3244.75 | 1 | 27.85 | 0 |
|  | 4 | 10800.00 | 10 | 10041.10 | 2 | 203.13 | 0 |
|  | 5 | 10800.00 | 10 | 8804.01 | 5 | 750.89 | 0 |
| 40 | 2 | 10800.00 | 10 | 10800.00 | 10 | 4.78 | 0 |
|  | 3 | 10800.00 | 10 | 10800.00 | 10 | 651.30 | 0 |
|  | 4 | 10800.00 | 10 | 10800.00 | 10 | 1538.69 | 0 |
|  | 5 | 10800.00 | 10 | 10800.00 | 10 | 7404.07 | 5 |

Table 1: CPU runtime in seconds. Comparison of MIP, CP Benders, and TSP Benders.

|  |  | (C1) |  | (C2) |  | (C3) |  |
|---|---|---|---|---|---|---|---|
| n | m | Avg Runtime | Avg # of Iter | Avg Runtime | Avg # of Iter | Avg Runtime | Avg # of Iter |
| 10 | 2 | 0.24 | 2.1 | 0.12 | 2.1 | 0.12 | 2.1 |
|  | 3 | 0.25 | 1.9 | 0.22 | 1.9 | 0.26 | 1.9 |
|  | 4 | 0.42 | 1.8 | 0.42 | 1.8 | 0.40 | 1.7 |
|  | 5 | 0.56 | 1.5 | 0.45 | 1.4 | 0.47 | 1.4 |
| 20 | 2 | 1.15 | 4.5 | 1.08 | 4.5 | 1.09 | 4.5 |
|  | 3 | 2.06 | 2.7 | 2.02 | 2.7 | 2.05 | 2.7 |
|  | 4 | 56.87 | 3.9 | 53.12 | 3.9 | 70.06 | 3.9 |
|  | 5 | 153.33 | 4.4 | 122.46 | 4.0 | 124.30 | 3.8 |
| 30 | 2 | 2.17 | 4.4 | 2.15 | 4.4 | 2.28 | 4.4 |
|  | 3 | 29.31 | 5.4 | 27.85 | 5.2 | 30.52 | 5.2 |
|  | 4 | 224.67 | 4.9 | 203.13 | 4.5 | 222.83 | 4.7 |
|  | 5 | 784.39 | 5.6 | 750.89 | 5.3 | 817.22 | 5.3 |

Table 2: CPU runtime in seconds. Comparison of different cuts on the TSP-Benders model.

the effort required in the master problem at producing assignments and enable the model to generate feasible schedules faster. Whether the method chosen is to allow the MIP model to solve with an optimality gap tolerance in mind or to solve the master problem through some other heuristic is to be determined.

The change to the Benders decomposition master problem means that the master problem would no longer act as a true lower bound. However, if the optimality gap suggestion is the method of choice, it could be guaranteed that the solution found would be within that chosen gap of optimal. Experimental results from large instances show that the master problem often spends most of time proving optimality or obtaining optimality with a gap of less than 2%. If the master problem was allowed to solve until it has proven it is within 2% of optimal, the Benders approach would be able to obtain solutions much faster. What is of concern however, is whether prematurely solving the master problem with this gap is detrimental to the quality of a solution if optimality is not hard to find or prove for some problems.

Another extension to this work is to test the performance of branch-and-check (Thorsteinsson 2001) on the PMSP with setup times. Beck (2010) identified that using branch-and-check, on problems with proportionately larger master problems in comparison to subproblems, is better than logic-based Benders. The Benders decomposition could benefit from solving the subproblem more often and create feasible schedules earlier.

## 6 Conclusion

In this paper, we presented a logic-based Benders decomposition approach to minimize the makespan of an unrelated parallel machine scheduling problem with sequence and machine dependent setup times. A MIP model was defined to solve for the assignment of jobs to machines and produce a lower bound on the achievable makespan of the problem. Two subproblems, based on a CP and TSP solver, were implemented to find optimal schedules for the assignment of jobs on each individual machine. The computational results show that the cooperation of MIP and TSP can effectively find optimal solutions. We are able to solve instances four times larger than what was previously possible using a MIP formulation found in the literature and obtain optimal solutions on problems of the same size up to five orders of magnitude faster. Although finding optimal solutions was the goal of the paper, possible heuristics based on the Benders decomposition approach are proposed to solve larger instances.

## References

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.

Allahverdi, A.; Gupta, J.; and Aldowaisan, T. 1999. A review of scheduling research involving setup considerations. *Omega* 27(2):219–239.

Arnaout, J.; Rabadi, G.; and Musa, R. 2008. A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing* 1–9.

Baker, K. 1974. *Introduction to sequencing and scheduling*. John Wiley & Sons.

Beck, J. 2010. Checking-Up on Branch-and-Check. In *Proceeings of the Sixteenth International Conference of Principles and Practice of Constraint Programming (CP2010)*, 84–98. Springer.

Cheng, T., and Sin, C. 1990. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research* 47(3):271–292.

Chu, Y., and Xia, Q. 2005. A hybrid algorithm for a class of resource constrained scheduling problems. In *Proceedings of the 2nd Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 110–124. Springer.

Dunstall, S., and Wirth, A. 2005. Heuristic methods for the identical parallel machine flowtime problem with set-up times. *Computers & Operations Research* 32(9):2479–2491.

Focacci, F.; Laborie, P.; and Nuijten, W. 2000. Solving scheduling problems with setup times and alternative resources. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling*.

França, P.; Gendreau, M.; Laporte, G.; and Muller, F. 1996. A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *International Journal of Production Economics* 43(2-3):79–89.

Glass, C.; Potts, C.; and Shade, P. 1994. Unrelated parallel machine scheduling using local search. *Mathematical and Computer Modelling* 20(2):41–52.

Graham, R.; Lawler, E.; Lenstra, J.; and Kan, A. 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics* 5(2):287–326.

Graves, S. 1981. A review of production scheduling. *Operations Research* 29(4):646–675.

Guinet, A. 1991. Textile production systems: a succession of non-identical parallel processor shops. *The Journal of the Operational Research Society* 42(8):655–671.

Helal, M.; Rabadi, G.; and Al-Salem, A. 2006. A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. *International Journal of Operations Research* 3(3):182–192.

Hooker, J., and Ottosson, G. 2003. Logic-based Benders decomposition. *Mathematical Programming* 96(1):33–60.

Hooker, J. 1995. Verifying logic circuits by Benders decomposition. *Principles and Practice of Constraint Programming: The Newport Papers, MIT Press, Cambridge, MA* 267–288.

Hooker, J. 2005. A hybrid method for the planning and scheduling. *Constraints* 10(4):385–401.

Kurz, M., and Askin, R. 2001. Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research* 39(16):3747–3769.

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(2):277–288.

Mendes, A.; Muller, F.; França, P.; and Moscato, P. 2002. Comparing meta-heuristic approaches for parallel machine scheduling problems. *Production Planning & Control* 13(2):143–154.

Rabadi, G.; Moraga, R.; and Al-Salem, A. 2006. Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing* 17(1):85–97.

Thorsteinsson, E. 2001. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP2001)*, 16–30. Springer.