Decomposition Methods for the Parallel Machine Scheduling Problem with Setups*

Tony T. Tran, Arthur Araujo, and J. Christopher Beck
Department of Mechanical and Industrial Engineering
University of Toronto, Toronto, ON M5S 3G8
{tran,araujo,jcb}@mie.utoronto.ca

Abstract

We study the unrelated parallel machine scheduling problem with sequence and machine dependent setup times and the objective of makespan minimization. Two exact decomposition-based methods are proposed based on logic-based Benders decomposition and branch-and-check. These approaches are hybrid models that make use of a mixed integer programming master problem and a specialized solver for travelling salesman subproblems. The master problem is used to assign jobs to machines, while the subproblems find optimal schedules on each machine given the master problem assignments. Computational results show that the decomposition models are able to find optimal solutions up to four orders-of-magnitude faster than the existing state of the art as well as solve problems six times larger than an existing mixed integer programming model. We further investigate the solution quality versus run-time trade-off for large problem instances for which the optimal solutions cannot be found and proved in a reasonable time. We demonstrate that the branch-and-check hybrid algorithm is able to produce better schedules in less time than the state-of-the-art metaheuristic, while also providing an optimality gap.

1 Introduction

Many practical scheduling problems consist of alternative machines and sequence dependent setup times where a job must be assigned to one of a set of machines and a minimum time must elapse between consecutive jobs executed on the same machine. The time which must elapse depends on the machine and the pair of jobs being executed. This situation arises, for example, in chemical plants where reactors must be cleaned when changing from processing one mixture to another and the time required for cleaning depends upon the previously completed and subsequent jobs. If the preceding chemical affects the succeeding one, longer cleaning times may be required to ensure that the reactor is properly prepared. The products processed in the reverse order may require a shorter cleaning time because the process may not suffer the same contamination risks. Further examples can be found in the plastic, glass, paper and textile industries where setup times of significant length exist [3, 17].

We study the unrelated parallel machines scheduling problem (PMSP) with sequence and machine dependent setup times. In the PMSP, jobs must be assigned to one of a set of alternative resources and jobs assigned to the same resource have setup times defined as the time that must elapse between the end of one job and the start of the next job. The setup time is sequence and machine dependent in that the elapsed time between jobs will differ depending on the order and machine to which the pair of jobs is assigned. We assume that setup times follow the triangle inequality. We concern ourselves with minimizing the makespan; that is, the maximum completion time of a schedule. Using the three-field

*A preliminary version of this paper appeared in a refereed conference as [36]. The extensions in this paper include the formulation of a new state-of-the-art algorithm for the problem studied (branch-and-check), comparison against a new state-of-the-art MIP model published after [36], and a comprehensive experimental study comparing the decomposition methods against metaheuristics on large problems.

notation given by [19], this problem can be denoted as (R|sds|C$_{max}$). Here, R indicates that there are multiple unrelated machines in parallel, sds represents the sequence dependent setup time characteristic of the problem, and C$_{max}$ denotes that minimizing the makespan is our objective.

We develop two exact methods designed to solve the PMSP: a logic-based Benders decomposition (LBBD) approach and a branch-and-check approach. Both methods make use of the same problem decomposition but vary in the timing at which the components of the decomposition are solved. We demonstrate that the two approaches provide significant improvements over the state-of-the-art approaches to the PMSP, both in terms of finding and proving the optimality of solutions, and (for the branch-and-check approach) the identification of high-quality solutions in low computational time. For moderately sized problems, the decomposition models are able to find and prove optimal solutions up to four orders-of-magnitude faster than the existing mixed integer programming (MIP) formulation. Equivalently, problems six times larger can now be solved to proven optimality in a reasonable amount of time. Looking at large problems where finding and proving optimality is not currently possible, branch-and-check is able to find better solutions in less time than state-of-the-art metaheuristic algorithms while also providing a bound on solution quality that is generally unavailable with metaheuristics. Our experiments show that branch-and-check is able to consistently provide better makespans and can do so an order-of-magnitude faster.

The strength of logic-based Benders decomposition for scheduling problems has been shown in a number of papers over the past 15 years (see [30] for a survey). The LBBD development here should be seen as a continuation of this line of work with the primary novelty being the first application of LBBD to PMSP with sequence and machine dependent setup times and, as far as we are aware, the first integration with a specialized TSP solver to solve subproblems. More significantly in terms of novelty, there has been very little work on branch-and-check. The only rigorous comparison between branch-and-check and LBBD is found in a conference paper [10]. In this paper, for the first time, we apply branch-and-check to the PMSP with sequence and machine dependent setup times, resulting in substantially improved solving performance compared to LBBD and MIP models. More significantly, we demonstrate that branch-and-check, unlike LBBD and MIP, out-performs the state-of-the-art metaheuristic approaches on large problem instances.

2 Background

In this section, we define the PMSP with sequence and machine dependent setup times and review previous work.

2.1 Problem Definition

In the PMSP, a set of jobs, N, are to be scheduled on a set of machines, M, with the objective of minimizing the makespan. Each job has a processing time p_{ij} corresponding to the time required to process job j on machine i. The machines in this system are unrelated; that is, a job j can have a processing time greater than job k on one machine, but the reverse could be true on another machine. There are sequence and machine dependent setup times, s_{ijk}: the time that must elapse between the completion of job j and the start of job k if job j precedes job k on machine i. Each job j also has a setup time s_{i0j} if it is the first job to be processed on machine i. The setup times are assumed to follow the triangle inequality: $s_{ijk} \leq s_{ijl} + s_{ilk}$. The goal of the problem is to determine how to assign jobs to machines and then sequence these jobs on each machine to minimize the makespan.

To formally define the PMSP, we present the current state-of-the-art MIP model presented in [7].

$$
\begin{aligned}
\text{minimize} \quad & C_{max} && && (1)\\
\text{s.t.} \quad & \sum_{j\in N_0, j\neq k} \sum_{i\in M} x_{ijk} = 1 && k \in N && (2)\\
& \sum_{k\in N_0, j\neq k} \sum_{i\in M} x_{ijk} = 1 && j \in N && (3)\\
& \sum_{k\in N_0, k\neq j} x_{ijk} = \sum_{h\in N_0, h\neq j} x_{ihj} && j \in N, i \in M && (4)\\
& C_k - C_j + V(1 - x_{ijk}) \geq s_{ijk} + p_{ik} && j \in N_0, k \in N, j \neq k, i \in M && (5)\\
& \sum_{j\in N} x_{i0j} \leq 1 && i \in M && (6)\\
& C_0 = 0 && && (7)\\
& \sum_{j\in N_0, j\neq k} \sum_{k\in N} (s_{ijk} + p_{ik}) x_{ijk} = O_i && i \in M && (8)\\
& O_i \leq C_{max} && i \in M && (9)\\
& x_{ijk} \in \{0, 1\} && j \in N_0, k \in N_0, j \neq k, i \in M && (10)
\end{aligned}
$$

where

C_{max}:	Maximum completion time (makespan)
C_j:	Completion time of job j
O_i:	Latest completion time of jobs on machine i
x_{ijk}:	1 if job k is processed directly after job j on machine i
M:	Set of machines
N:	Set of jobs to schedule
N_0:	Set of jobs to schedule with an additional dummy node (indexed by 0)
V:	A large positive number

The MIP model is a mathematical representation of the scheduling problem. Two decision variables represent the decisions associated with a job: x_{ijk} and C_j. These decision variables will define the machine that the job is processed on, the sequence of processing of jobs on a machine, and the completion time of the job. Constraint (2) (resp. (3)) ensures that each job has a single predecessor (successor) on exactly one of the machines. Constraint (4) states that a job can only have a predecessor on a machine if it also has a successor on that same machine. Constraint (5) sets the completion times of each job such that if job j precedes job k on a machine i, the earliest time that job k can end must be greater than the completion time of job j plus the setup time from job j to job k and the processing time of job k. If job k is processed directly after job j on machine i, $1 - x_{ijk} = 0$ and the constraint becomes $C_k - C_j \geq s_{ijk} + p_{ik}$. If job k is not scheduled directly after job j on a machine, $1 - x_{ijk} = 1$ and the large V term makes the constraint non-binding. This constraint will ensure that a single valid sequence of jobs is scheduled on each machine and that none of the jobs' processing time overlap one another. Constraint (6) guarantees that only one job can be scheduled first on each machine. Constraint (7) sets the completion time of job 0, an auxiliary job used to enforce the start of a schedule, to zero. Constraint (8) calculates the makespan for each individual machine and constraint (9) links the individual machine makespan to the overall schedule makespan. Constraint (10) defines the decision variables as binary.

2.2 Related Work

The PMSP with sequence and machine dependent setup times (R|sds|C$_{max}$) is strongly NP-hard as the single machine scheduling problem with sequence dependent setup times (1|sds|C$_{max}$) is equivalent to a travelling salesman problem (TSP) [8].

2.2.1 Exact Approaches

The combinatorial complexity of the PMSP has resulted in little research in exact optimization methods. Until recently, the only MIP models to solve the PMSP with sequence and machine dependent setup times were minor variations of a model proposed by [20] and used in a number of papers as a straw man comparison to metaheuristic approaches [21, 32]. With modern hardware and commercial software, only small instances (e.g., 8 jobs and 4 machines or 9 jobs and 2 machines) could be solved and proved optimal using this model.

The MIP model presented above achieves substantial improvement with two modifications to the previous formulation [7]. The first change is to include Constraint (8) which exactly calculates the completion time on each individual machine based on the assignment and sequencing. The second addition is to use Constraints (7) and (8) to explicitly calculate the makespan for each machine rather than relying on the maximum C_i value. That is, in the previous model Constraints (7) and (8) were represented by $C_i \leq C_{max}, i \in M$. Although C_i is now not used for calculation of the makespan, it is still required for Constraint (5) to ensure that each machine will only have a single, non-overlapping sequence of jobs. These two changes improve the performance of the MIP so that it is able to solve problems up to four times larger and five orders-of-magnitude faster than the previous MIP models [7].

Other research on the PMSP with unrelated machines has examined variations of the problem studied here. An exact branch-and-bound algorithm was developed by [28] to minimize makespan for a problem with two parallel machines but without setup times. [33] looked at exact algorithms for the PMSP with machine and sequence dependent setup times and due dates, with the objective of minimizing a function of makespan and total tardiness. They compared their branch-and-bound algorithm against two MIP models and showed that their model is capable of optimally solving problems of size up to 30 jobs, significantly out-performing the MIP models known at the time.

Another related problem class is the min-max vehicle routing problems (VRP). In the VRP, multiple vehicles must travel between nodes to service all customers. Generally, VRPs correspond to the PMSPs with identical machines and 0 processing times. [37] presented a branch-and-cut algorithm for the VRP, but showed that even with only two vehicles, the algorithm is unable to solve most instances tested. They improved their results via a column generation heuristic, but reported high duality gaps (10% or more) on many of the test instances. For the class of heterogeneous vehicles, [9] provide a survey where they described various different VRP problems and solution approaches.

The state of current research regarding solving PMSPs with machine and sequence dependent setup times to optimality is limited to problems with fewer than 40 jobs. Even for problems that consist of 40 jobs and 5 machines, the current best MIP model is only able to solve 5 of the 30 tested benchmark instances in less than one hour [7].

2.2.2 Heuristic Approaches

There has been comparatively more work on heuristic approaches to the PMSP with sequence and machine dependent setup times. Early work by [18] compared a genetic algorithm, simulated annealing, and tabu search. Empirical results indicated that the tabu search approach was capable of finding the best solutions in short periods of time (20 seconds), however the performance comparison for a larger time limit (100 seconds) shows no clear distinction among the three methods.

[2] developed a *partitioning* heuristic designed for large instances of the PMSP with setup times. The approach makes use of a constructive and improvement heuristic which assigns jobs to machines coupled with a TSP-like heuristic that subsequently sequences the jobs. [32] presented a metaheuristic called Meta-RaPS. The heuristic is a modified COMSOAL (Computer Method of Sequencing Operations for Assembly Lines) approach [5] with the goal of minimizing makespan. Direct comparisons were made between Meta-RaPS and the partitioning heuristic on problems up to 120 jobs and 12 machines where Meta-RaPS was found to be 10% better in terms of solution quality in some cases.

[21] developed a tabu search to solve the same problem and also compared their model to the partitioning heuristic. The tabu search was found to be up to 8% better than the partitioning heuristic on the same sized instances.

Most recently, the state-of-the-art heuristic approach to PMSP with sequence and machine dependent setup times is ant colony optimization (ACO) [6]. All four of the above heuristic approaches were tested on instances up to size of 120 jobs and 8 machines where quality was defined as the difference between the makespan found and a simple lower bound. ACO was found to produce schedules with makespans within 1% to 7% deviation of the lower bound with Meta-RaPS close behind. The Scheduling Research Virtual Center website (http://schedulingresearch.com/) contains many of the problem instances and performance data for these metaheuristic models.

Heuristic approaches have also been applied to closely related problems. [16] proposed a two-phase algorithm based on constraint programming (CP) to heuristically optimize a combination of the makespan and the sum of setup times. Jobs consist of multiple activities and, unlike our problem, 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 the makespan must not exceed the makespan found in the first phase. They tested their model on problems of up to 16 jobs, each consisting of 12 activities and 16 machines.

An unrelated PMSP with sequence and dependent setup times and the objective of minimizing tardiness was studied by [29]. A tabu search was shown to be able to find (but not prove) optimal solutions on more than 50% of the small problems (up to 10 jobs and 4 machines) while performing well on larger instances.

3 Two Decomposition Methods

In this section, we present the two decomposition models for our problem: a logic-based Benders decomposition and the closely related branch-and-check decomposition. Both decompositions make use of the same algorithm components (master problem, subproblem, and cut), but differ in when the subproblems are solved. We first provide an overview of the two decompositions followed by detailed definitions of the algorithm components.

3.1 Logic-Based Benders Decomposition

Logic-based Benders decomposition (LBBD) is a generalization of classical Benders decomposition [11] developed by Hooker [22, 23, 26]. It has been applied to a wide range of combinatorial optimization problems including the verification of logic circuits [22], planning and scheduling [24, 25, 12, 4], stochastic and deterministic location/fleet management [14, 15], and queue design and control [34]. In particular, [25] uses LBBD to solve the PMSP problem without setup times with the objective of minimizing cost, makespan, or total tardiness.

We present the notation and explanation due to [26]. To model a problem for LBBD, one must first partition the decision variables into two vectors x and y. In general, we can view the problem as,

$$\begin{aligned} \text{minimize} \quad & f(x,y) && (11)\\ \text{s.t.} \quad & (x,y) \in S, && (12)\\ & x \in D_x,\ y \in D_y && (13) \end{aligned}$$

Here, f is a real-valued function, S is the feasible set (generally defined by a collection of constraints), and D_x and D_y are the domains of x and y respectively. The problem is split into constraints solely involving the x or master problem variables and constraints that mix the x and y, subproblem variables. Since the master problem only considers the x variables, the feasible set S is relaxed and is denoted as $\bar{S}$. Unlike classical Benders decomposition, there are no structural restrictions (e.g., linearity) on the different components of the decomposition.

Formally, the master problem can be defined as follows:

$$\begin{aligned} \text{minimize} \quad & z && (14)\\ \text{s.t.} \quad & x \in \bar{S}, && (15)\\ & z \geq \beta_{x^k}(x), \qquad k = 1, ..., K && (16)\\ & x \in D_x. && (17) \end{aligned}$$

where z is a real-valued decision variable, $\bar{S}$ is a relaxation of S, and $\beta_{x^k}(x)$ is a *Benders* cut on the objective function $f(x, y)$ found when fixing values of x to x^k. Constraints (16) are derived from solving the subproblem and $x^1, ..., x^K$ are the values of the x variables in master problem solutions found during previous iterations.

The subproblem for iteration k is:

$$\begin{aligned} \text{minimize} \quad & f(x^k, y) && (18)\\ \text{s.t.} \quad & (x^k, y) \in S, && (19)\\ & y \in D_y. && (20) \end{aligned}$$

Solving an LBBD model consists of iteratively solving the master problem and the subproblems until convergence. The master problem is solved to optimality, producing solution x^k with cost z^k in iteration k. That solution is then used to formulate one or more subproblems which are each solved, producing bounding functions (i.e., the *Benders* cuts) on z. If the k-th master problem solution satisfies all the new bounding functions obtained in iteration 1 to k, the process has converged to a globally optimal solution (i.e., $z^k = f(x^k, y^k)$, where y^k is the subproblem solution). Otherwise, the master problem is solved again, and the iterations continue. Under reasonable conditions on the Benders cuts, it can be shown that the process converges to an optimal solution in a finite number of iterations [13] (see Section 4.4 below for details).

While LBBD is typically presented as above, one core idea is that the master problem is a relaxation of the global problem. It is not necessarily the case that the relaxation must be achieved by partitioning the variables and ignoring the y variables in the master problem. Instead, one can form $\bar{S}$ by relaxing another aspect of the problem. For example, when y are integer, we can ignore their integrality in the master problem. In this way, the master problem is easier to solve than the original problem, but has a tighter representation than if the y variables were completely ignored in the master problem. In Section 4.2, we present such a master problem formulation that includes both the x and y variables.

3.2 Branch-and-Check

In branch-and-check [35], the subproblem is solved during the branch-and-cut search of the master problem whenever a *feasible* master problem solution is found. Thus, when referring to branch-and-check, we define x^k and z^k as the k^{th} feasible solution and objective value found by the master problem, respectively. If all subproblems have an optimal solution which is less than or equal to z^k, a globally feasible solution has been found. Otherwise, a Benders cut is generated and added to the master problem model. In either case, the branch-and-cut search of the master problem then continues until it is proved that the last globally feasible solution found is optimal.

The definition of logic-based Benders decomposition [23] was general enough to encompass both LBBD and branch-and-check as defined above. In fact, Hooker's definition allows subproblem solving to be triggered without a feasible master solution by employing rounding heuristics to establish values for x^k at a node in the master branch-and-cut search. [35] subsequently coined the term *branch-and-check* and more explicitly developed the ideas in [23]. However, in his empirical analysis, [35] did not investigate the full branch-and-check idea: the master solver was stopped at each feasible solution, the subproblem(s) were solved and, if necessary, Benders cuts were added. However, the master branch-and-cut search was then *restarted* rather than continued.

[10] performed the first systematic empirical comparison of LBBD and branch-and-check, demonstrating that their relative performance depends on the difficulty of solving the master problem as compared with the subproblems. On the problems tested, when subproblems were easy to solve compared to the master problem, branch-and-check was superior to LBBD. In contrast, LBBD was superior when solving a subproblem was hard compared to solving the master problem. This conclusion led us to consider branch-and-check as a promising approach to the PMSP with setups as subproblems can be solved very quickly with dedicated solvers, as described below.

4 LBBD and Branch-and-Check for PMSP with Setups

In this section, we provide the detailed model of the PMSP with setups using LBBD and branch-and-check. In particular, we define the master problem, the subproblem, a Benders cut, and criteria for LBBD to find sub-optimal solutions before an optimal solution is found.

4.1 Mixed Integer Programming Model

To fit the proposed decomposition into the LBBD framework presented in Section 3.1, we first introduce an alternative MIP model for the global problem.

$$\begin{aligned}
\text{minimize} \quad & C_{max} && && (21)\\
\text{s.t.} \quad & \sum_{j \in N} x_{ij} p_{ij} + \xi_i \le C_{max}, && i \in M && (22)\\
& \sum_{i \in M} x_{ij} = 1, && j \in N && (23)\\
& x_{i0} = 1, && i \in M && (24)\\
& \xi_i = \sum_{j \in N_0} \sum_{k \in N_0} y_{ijk} s_{ijk}, && i \in M && (25)\\
& x_{ik} = \sum_{j \in N_0} y_{ijk}, && k \in N_0;\ i \in M && (26)\\
& x_{ij} = \sum_{k \in N_0} y_{ijk}, && j \in N_0;\ i \in M && (27)\\
& C_k - C_j + V(1 - y_{ijk}) \ge s_{ijk} + p_{ik} && j \in N_0, k \in N, i \in M && (28)\\
& C_0 = 0, && && (29)\\
& x_{ij} \in \{0; 1\}, && j \in N;\ i \in M && (30)\\
& y_{ijk} \in \{0; 1\}, && j, k \in N_0;\ i \in M && (31)
\end{aligned}$$

where

C_{max}:	Maximum completion time
ξ_i:	Total setup time incurred 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
C_j:	Completion time of job j

The makespan on each machine is defined in constraint (22) as the summation of processing times for all jobs assigned to that machine plus the total setup time. Constraint (23) ensures that each job is assigned to exactly one machine. Constraint (24) places the dummy job (which signifies the start and end of a sequence of jobs on a machine) on each machine. Constraint (25) assigns the setup time of a machine i, ξ_i, to be the additional time required from the sequencing of jobs, y_{ijk}, and their respective setup times, s_{ijk}. Constraints (26) and (27) state that if a job j is assigned to a machine i, then there must be exactly one other job that precedes and one job that succeeds job j on machine i. Constraint (28) is the same constraint as Constraint (5) used to set the completion time of jobs in a sequence. Constraint (29) places the dummy job at time 0. Finally, Constraints (30) and (31) set x_{ij} and y_{ijk} variables as binary decision variables.

4.2 Assignment Master Problem

The MIP formulation of the master problem is a relaxation of the MIP in the previous section. In this relaxation, jobs are assigned to machines, but instead of requiring a single sequence of jobs on each machine, multiple, disjoint sequences are allowed, which together include all jobs assigned to the machine. The relaxation essentially removes the decision variable C_j and Constraints (28) and (29) from the MIP model. However, instead of removing y_{ijk} from the model completely, the decision variable is relaxed to be any real-valued number between 0 and 1. By including y_{ijk} in the master problem, a tighter formulation that partially takes into account setup times is achieved. The master problem is as follows:

$$\begin{aligned}
\text{minimize} \quad & C_{max} && && (32)\\
\text{s.t.} \quad & \text{Constraints (22) to (27)} && && \\
& \textit{cuts} && && (33)\\
& x_{ij} \in \{0;1\}, && j \in N;\ i \in M && (34)\\
& 0 \leq y_{ijk} \leq 1, && j,k \in N_0;\ i \in M && (35)
\end{aligned}$$

where

x_{ij}:	1 if job j is processed on machine i
y_{ijk}:	Relaxed representation of job k being processed directly after job j on machine i

The master problem includes Constraints (22) to (27) from the MIP model. However, instead of ξ_i equating to the total setup time of sequencing jobs on a machine i, a relaxation is applied so that it is now a lower bound on the actual setup times. The relaxation allows smaller, independent sequences instead of a single sequence of jobs on a machine. From a TSP perspective, the master problem allows sub-tours where each job has a predecessor and successor, but multiple sub-cycles may exist. For example, given jobs j_1, j_2, j_3, j_4, and j_5, Constraints (26) and (27) do not prevent sequences [*start* - j_1 - j_2 - j_3 - *end*] and [j_4 - j_5 - j_4 - j_5...]. Constraint (33) is the Benders cut added to the master problem each time a subproblem results in a higher makespan than that of its master problem solution. The cuts are defined below in Section 4.4. The set of cuts is empty in the first iteration of the master problem.

4.3 Sequencing Subproblems

The solution to the master problem defines the subproblems. Let x_{ij}^{h*}, y_{ijk}^{h*}, and C_{max}^{h*} be the solution found from the master problem at iteration h. The x_{ij}^{h*} values provide an assignment of each job to one of the $|M|$ machines. Since the sequence from the master problem may not be feasible due to the presence of multiple cycles on a single machine, the y_{ijk}^{h*} values are ignored in the subproblem. Each time the subproblems must be solved, whether for logic-based Benders decomposition or for branch-and-check, we create $|M|$ separate problems each representing one machine and containing only the jobs assigned to that machine (i.e., for machine i', only jobs j where $x_{i'j}^{h*} = 1$). Given a fixed assignment, the sequence of jobs on one machine does not affect another machine and so we are able to solve each machine subproblem independently.

A subproblem must sequence the assigned jobs with the goal of minimizing the makespan. This problem is a single-machine scheduling problem with sequence-dependent setup times, $(1|sds|C_{max})$, which can be modelled as an asymmetric travelling salesman problem (ATSP). In an ATSP, we are given a complete graph $G = (V, E)$ with non-negative weights on each edge. Here, V is the set of nodes and E is the set of edges. The objective is to find a Hamiltonian cycle (a tour that passes through all the nodes) of G with the minimal sum of edge weights.

Given a set of jobs $N_i^h = \{j : x_{ij}^{h*} = 1\}$ (the jobs assigned to machine i from iteration h of the master problem), we can represent the $(1|sds|C_{max})$ as an ATSP with $|N_i^h| + 1$ nodes. The nodes represent jobs and the weight of edge (j, k) is the sum of the processing time of job j, p_{ij}, and the transition time from job j to job k on machine i, s_{ijk}. The weight of the edge from the dummy node to job j is equal to the setup time of job j if it were the first job to be processed on a machine, s_{i0j}. An edge from job j to the dummy node has weight of the processing time of job j, p_{ij}. A solution to the ATSP corresponds to a minimal length sequence of jobs.

The ATSP representation is shown in Figure 1. It can be seen that if the order of jobs to be processed is 1, 3 then 2, the distance travelled would be, $s_{i01} + p_{i1} + s_{i13} + p_{i3} + s_{i32} + p_{i2}$. This tour distance is equal to the makespan of processing jobs in that order.

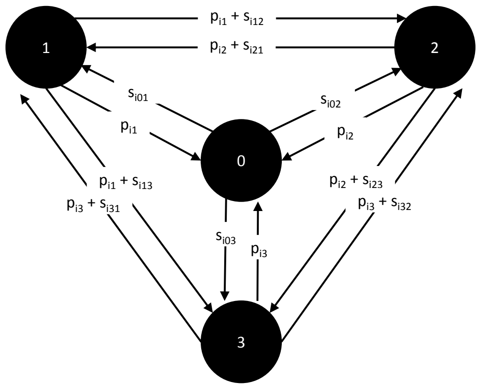


Figure 1: The TSP representation of a subproblem.

The representation shown above allows us to take any $(1|sds|C_{max})$ subproblem and create an ATSP. Hence, the subproblems are reformulated in terms of an ATSP. We then convert the problem to be a symmetric TSP in order to use existing (symmetric) TSP solvers. We use the three-node transformation due to [27]: we replace every node j by three nodes: j^-, j^0, and j^+. This results in a graph G' with $3(|N_i| + 1)$ nodes. The edges (j^-, j^0) and (j^0, j^+) have a weight of 0 and edges (j^+, k^-) are equal to the weight on arc (j, k) from the original graph G.

4.4 Cuts

If the makespan of a machine found in the subproblem is less than or equal to the master problem's makespan, then this solution is feasible with respect to the master problem and no cuts are added. Otherwise, a cut is created. The master problem is then updated with the cuts from each such subproblem.

To define the cut, we introduce the value $maxPre_{hj}$, the maximum setup time if job j directly succeeds another job that is assigned in the master problem to the same machine in iteration h. That is,

$$maxPre_{hj} = \max_{k \in N_{i'}^h; k \neq j} (s_{i'kj}),$$

where i' is the machine that job j was assigned to in the master problem in iteration h. We define $\theta_{hi'j}$ to be the sum of the processing time of job j on machine i' and $maxPre_{hj}$.

$$\theta_{hi'j} = p_{i'j} + maxPre_{hj}.$$

The proposed cut is then

$$C_{max} \geq C^{hi'*}_{max} - \sum_{j \in N^h_{i'}} (1 - x_{i'j})\theta_{hi'j}.$$

Here, C_{max} is the makespan variable in the master problem and $C^{hi'*}_{max}$ is the makespan found in iteration h for machine i'. The cut places a lower bound on the makespan in future iterations, depending on the jobs that are assigned. 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 $\sum_{j \in N^h_{i'}} (1 - x_{i'j}) = 0$ and the makespan of the subproblem becomes a new lower bound on C_{max}. When a different assignment is made and at least one of the $x_{i'j}$ variables that previously had a value of 1 is 0, the cut enforces that the makespan in a future iteration must be bounded by the makespan found in the subproblem reduced by the corresponding θ_{hij} value(s).

A valid cut must adhere to two conditions [13]: i) it must remove the current solution from the master problem and ii) it must not eliminate any globally optimal solutions. The first condition is easily seen. If a subsequent iteration assigns exactly the same set of jobs to machine i, the master problem must increase the value of the makespan variable. Otherwise, a change in assignment to machine i is necessary. In either case, the solution of iteration h is removed from the master problem space.

As the second condition is slightly more involved, we prove the following theorem:

Theorem 1. *The proposed cut does not remove any globally optimal solutions.*

Proof. Proof We proceed by assuming that there exists a globally optimal schedule violating the cut and then showing a contradiction.

Given a set of jobs, N, assigned to machine i in the current iteration, let C_N be the optimal makespan on machine i. Suppose there exists a globally optimal schedule that violates the cut generated for machine i in iteration h. Let N^* be the set of jobs assigned to i by this globally optimal schedule and let C_{N^*} be the makespan of machine i in the global schedule. As the globally optimal schedule does not assign the jobs $\hat{N} := N - N^*$ from N to machine i and it violates the cut under our initial assumption, the following property holds:

$$C_{N^*} < C_N - \sum_{j \in \hat{N}} \theta_{hij}. \tag{36}$$

Given the schedule corresponding to C_{N^*}, it is possible to construct a reduced schedule by removing the jobs in $N^* - N$. This reduced schedule consists of the jobs $\bar{N} := N \cap N^*$ that are arranged in the same order as in the globally optimal solution. Let $C_{\bar{N}}$ be the makespan of machine i in the reduced schedule. As the setup times satisfy the triangle inequality, $C_{\bar{N}}$ is smaller than or equal to C_{N^*}. Hence, the reduced schedule also violates the cut:

$$C_{\bar{N}} < C_N - \sum_{j \in \hat{N}} \theta_{hij}. \tag{37}$$

To prove a contradiction, we need only to show that equation (37) cannot be true and therefore equation (36) is also not true since $C_{\bar{N}} \leq C_{N^*}$.

To show that equation (37) cannot be true, the reduced schedule is extended to a schedule containing all of N jobs by placing each job in $\hat{N}$, one by one, at the end of the reduced schedule. As $maxPre_{hj}$ is the maximum setup time over all jobs in N, for job j, the makespan of the resulting schedule, C'_N, must satisfy:

$$C'_N \leq C_{\bar{N}} + \sum_{j \in \hat{N}} (p_{ij} + MaxPre_{hj}) = C_{\bar{N}} + \sum_{j \in \hat{N}} \theta_{hij}.$$

Rearranging and using the assumption that C_N is the minimal makespan for the problem with the full set of N jobs:

$$C_{\bar{N}} \geq C'_N - \sum_{j \in \hat{N}} \theta_{hij} \geq C_N - \sum_{j \in \hat{N}} \theta_{hij}.$$

This contradicts equation (37) and therefore, the cut does not remove any optimal solutions from the global search space. □

□

Note that there is no requirement that the set of jobs assigned to machine i is part of an optimal solution to the master problem. Any feasible master solution that cannot be extended to a solution to the subproblem will produce a cut that removes the master solution and does not remove any globally optimal schedules. Thus, the cut is valid for branch-and-check as well as LBBD.

4.5 Finding Sub-optimal Solutions

Since the PMSP with setup times is NP-hard, it is not always possible to find optimal solutions in a reasonable time (unless $P = NP$). In the general formulation of LBBD, the first globally feasible solution is a globally optimal solution, meaning that for large problems, LBBD may not be able to find *any* feasible solutions in a reasonable time. Branch-and-check partially addresses this issue because globally feasible solutions can be found during the master branch-and-check search when a feasible but not optimal master solution is found to be consistent with each subproblem. We can more fully address this issue by converting any feasible master solution to a feasible global solution. In this section, we show two methods for LBBD and one for branch-and-check to obtain globally feasible solutions.

The first method can be used in both LBBD and branch-and-check. A feasible master problem solution assigns each job to a machine, forming the subproblems. If we ignore the C_{max} value in the master problem, then the maximum makespan over all subproblems is globally feasible. Therefore, we can find a feasible global solution for each feasible master solution. The global solution may not improve the makespan compared to previous iterations. However, we simply keep track of the best schedule found so far. In this way, a feasible schedule for the global PMSP exists once the first feasible master solution is found and the subproblems are solved.

For LBBD, maintaining the best solution found so far also provides a second stopping criteria. As described above in Section 3.1, LBBD terminates when an optimal master solution can be extended to a solution with the same or better makespan in each of the subproblems. In addition, LBBD can terminate if an optimal master solution is found whose makespan is equal to the best global solution found so far. The optimal master solution proves that no better global solution exists.

Another method which can be used to obtain a sub-optimal solution for LBBD is to require that the master problem is not solved to optimality, but to within some optimality gap. Doing so reduces the effort required in the master problem and enables the model to generate globally feasible schedules faster. Since LBBD normally requires the master problem to be solved to optimality at least once prior to a feasible solution being available, it may not be possible to find even a feasible global solution within a restricted time limit. Thus, specifying a non-zero optimality gap may enable sub-optimal global solutions to be found, but with a trade-off of no longer guaranteeing an optimal solution. The optimality gap is a parameter chosen by the user. The smaller the gap, the more difficult the master problem is to solve.

Note that even though the chosen optimality gap is used to stop the master search early, it is a valid bound on the quality of the global solution. To see this, consider a master-feasible solution, s, with a master optimality gap, δ. As the master problem is a relaxation of the global problem, if s is globally feasible, then it must have a global optimality gap of no more than δ because the master dual bound is a valid global dual bound.

5 Numerical Study

In this section, we test our decomposition models against the MIP model presented in Section 2.1 and several state-of-the-art metaheuristics. Two sets of experiments are performed. In the first set, we focus on finding provably optimal solutions and compare run-times with the current best MIP model [7]. The second set of experiments examines finding high quality solutions to larger problems, where finding provably optimal solutions is beyond the computational limits of existing techniques. We compare against state-of-the-art metaheuristic algorithms: an ant colony optimization [32] and a tabu search algorithm [21].

5.1 Finding and Proving Optimal Schedules

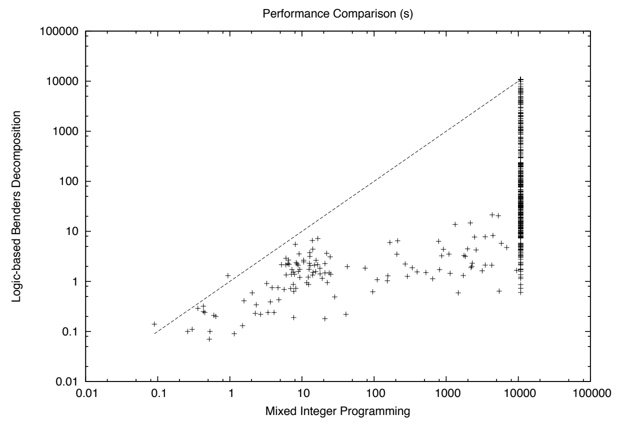
We test our models on a Intel Core i7 3.00 GHz CPU (in 64 bit mode) with 2 MB cache per core, 12 GB of main memory, running Linux. To solve the MIP model and master problem, SCIP 3.0.1 [1] is used. We chose SCIP over commercial solvers such as CPLEX and Gurobi due to SCIP being an open source solver that gives better insight into the exact way the problem is solved. Furthermore, the architecture of SCIP lends itself well to the framework of LBBD and branch-and-check which requires heavy use of solver callbacks. The TSP solver used in the subproblems is Concorde, available at http://www.math.uwaterloo.ca/tsp/concorde.html.

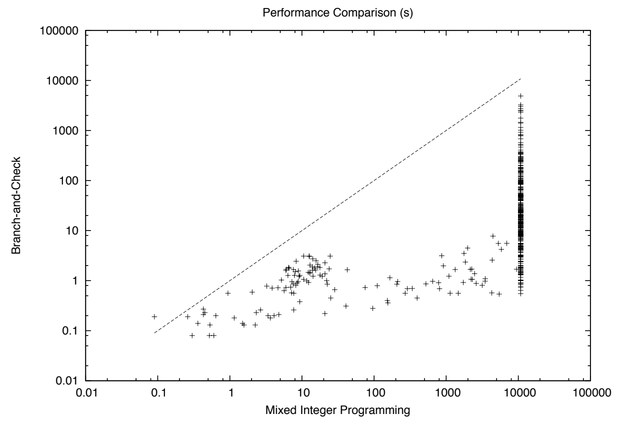
Experiments were run for problem instances of between 10 and 60 jobs, in increments of 10 jobs. Between 2 and 5 machines were tested for each job size. Each of these combinations has 20 instances for a total of 480 instances. A time limit of 3 hours was used for each instance. Processing times for each machine-job pair were generated from a uniform distribution between 5 and 200. 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 coordinates along the x and y axis are chosen using a uniform distribution between [0,50]. The setup times are the Manhattan distances from one job's coordinates to the others. Distances between the second set of coordinates are used to provide asymmetric setup times by linearly scaling a distance of 0 to the lower bound of the setup time distribution and 100 to the upper bound. For example, job 1 and job 2 would be given coordinates χ_{1a}, χ_{1b}, ψ_{1a}, ψ_{1b}, χ_{2a}, χ_{2b}, ψ_{2a}, and ψ_{2b}. If $[l, u]$ represents the lower and upper bound of setup times we wish to consider, setup time from job 1 (2) to job 2 (1) would then be $l + \frac{(u-l)}{100}\left[|\chi_{1a} - \chi_{2a}| + |\psi_{1a} - \psi_{2a}|\right]$ $(l + \frac{(u-l)}{100}\left[|\chi_{1b} - \chi_{2b}| + |\psi_{1b} - \psi_{2b}|\right])$. We set our setup times to be between 25 and 50.

Figure 2 shows scatter-plots of the pairwise comparisons between each of the three methods: MIP, LBBD, and branch-and-check. Each point corresponds to a single problem instance. For these results, the times at which an optimal solution were found and proved were recorded. Where the solving timed out, three hours was used. The format of the graphs means that points below the $y = x$ line show superior performance of the approach corresponding to the y-axis. Note the log-scales on all three plots.

We see that the decomposition models are both substantial improvements over the MIP model. The mass of points at the right side of the graphs indicates problems for which the MIP model was unable to find and prove an optimal solution within three hours. For all problem instances, except for one or two which are solved in less than one second, the decomposition methods out-perform MIP, sometimes by four orders-of-magnitude. The final graph shows that branch-and-check does not always outperform LBBD, but it generally does, especially on harder problems. The cases where LBBD is superior tend to be those instances which are solved within minutes.

Table 1 shows more detailed results for each of the problem sizes. The decomposition methods provide a vast improvement over the MIP model which can only solve instances with up to $|N| = 20$. Even at only 20 jobs, with three or more machines the MIP solver is not able to find and prove optimality in all instances. No instances with 20 jobs and five machines are solved using MIP. Between the two decomposition methods, branch-and-check is able to solve more instances and can do so up to an order-of-magnitude faster.





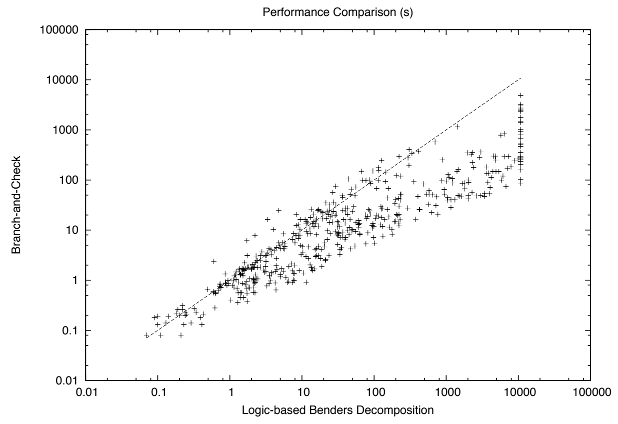


Figure 2: Pairwise run-time comparisons for each problem instance.

		MIP		**LBBD**		**Branch-and-Check**	
|N|	|M|	Avg Run-time	# uns.	Avg Run-time	# uns.	Avg Run-time	# uns.
10	2	2.58	0	0.22	0	**0.17**	0
	3	5.97	0	1.2	0	**0.74**	0
	4	13.67	0	1.96	0	**1.34**	0
	5	14.54	0	2.83	0	**2.14**	0
20	2	1070.57	0	1.35	0	**0.66**	0
	3	4166.72	5	6.58	0	**1.28**	0
	4	6262.64	8	15.40	0	**4.38**	0
	5	8134.65	11	48.22	0	**9.83**	0
30	2	7653.75	11	3.52	0	**1.20**	0
	3	9774.16	16	36.03	0	**5.41**	0
	4	10800.00	20	187.51	0	**21.30**	0
	5	10800.00	20	502.10	0	**62.32**	0
40	2	10800.00	20	11.49	0	**4.53**	0
	3	10800.00	20	61.28	0	**11.41**	0
	4	10800.00	20	521.64	0	**50.14**	0
	5	10800.00	20	3197.31	0	**159.19**	0
50	2	10800.00	20	22.59	0	**7.93**	0
	3	10800.00	20	413.14	0	**26.65**	0
	4	10800.00	20	2539.14	1	**101.26**	0
	5	10800.00	20	6112.01	9	**590.11**	0
60	2	10800.00	20	50.16	0	**17.99**	0
	3	10800.00	20	1044.22	0	**42.95**	0
	4	10800.00	20	3654.87	4	**261.99**	0
	5	10800.00	20	8056.22	10	**1253.43**	0

Table 1: Comparison of MIP, LBBD, and branch-and-check: Mean CPU run-time in seconds and the number of unsolved instances.

5.2 Finding Good Sub-optimal Schedules

In the second set of experiments, we test instances with many more jobs and machines in order to compare our decomposition methods against the state-of-the-art metaheuristics for PMSP with setups [21, 32]. Instances of up to 120 jobs were tested in previous papers using tabu search (TS) and ant colony optimization (ACO). As we can see from our previous experiment, the problem becomes significantly harder once we have increased the problem size to more than 60 jobs.

We run our models on the same test instances used to evaluate the metaheuristic algorithms; these instances were supplied by Dr. Ghaith Rabadi. The instances have processing time and setup times ranging from 50 to 100. As in [6], we vary the number of jobs evaluated between 40 and 120 with increments of 20 jobs. The number of machines used varies between 2 and 8 in increments of 2. However, consistent with the literature [6], we only considered combinations of machines and jobs when the number of jobs is at least 15 times larger than the number of machines. We test 15 instances for each combination that fits our job to machine ratio criteria.

Some of the performance information of the metaheuristics on these instances can be found at *Scheduling Research* [31]. While we do not have the run-time for each instance, the website provides the makespan found by each approach on each instance and the mean run-time over each machine-job problem size pair when using ACO. As a result, we are able to compare solution quality exactly, but run-time only imprecisely. Furthermore, as the hardware used for the metaheuristics is different than that used here, the run time results need to be interpreted with caution. The ACO algorithm was tested on a computer running Windows XP with a Pentium 4 processor and 2 Megabytes of RAM.

Following the custom in the metaheuristic literature, we compare the sub-optimal solutions to a lower bound (LB) calculated for each problem instance. The lower bound used is the same as that used in the work on ACO and is calculated using the following equations:

$$\begin{aligned} LB1 &= \tfrac{1}{|M|}\textstyle\sum_{j\in N}\min_{i\in M;k\in N}[p_{jk}+s_{ijk}] \\ LB2 &= \max_{j\in N}\{\min_{i\in M;k\in N}[p_{jk}+s_{ijk}]\} \\ LB &= \max(LB1, LB2) \end{aligned}$$

The deviation of the schedule (or mean relative error) from the lower bound is $\Delta = \frac{C_{max}-LB}{LB} \times 100\%$; a lower Δ is better.

We run the branch-and-check algorithm with exactly the same parameters as above. In contrast, for LBBD we specify a 2% optimality gap. If a solution with a 2% gap is not found, the algorithm returns the best available solution after timeout. We found that MIP and LBBD with an optimality gap of 0% were only able to find *feasible* solutions within 3 hours for some of the two-machine problems and none of the larger problems. As such, we do not report the performance of these two models.

Table 2 shows the performance comparison based on run-time and solution quality. After three hours, if the optimal solution has not been found and proved, the best solution found so far is used. For tabu search (TS) we only report solution quality (Δ) as the run-time was not available to us. For ACO and LBBD both solution quality and mean run-time are reported. Finally, for branch-and-check we report the solution quality at a number of time points.

LBBD with a 2% optimality gap appears competitive with ACO and is better than TS in terms of solution quality, finding better quality solutions on average in 7 of the 12 problem sizes. The run-time results as compared to ACO are a bit more mixed, especially on larger problems. In general, LBBD with a 2% optimality gap finds better solutions, but at the cost of a longer run-time.

Branch-and-check in contrast finds better quality solutions than ACO and TS on average with a run-time of 600 seconds over all problem sizes.

Table 2 also presents, in parentheses, the optimality gap found by branch-and-check at each time point. That is, we calculate $\Delta' = \frac{C_{max}-LB'}{LB'} \times 100\%$, where LB' is the dual bound for the master problem. With the exception of the thirty second time point for 60 jobs and 4 machines, the optimality gap provided by branch-and-check is significantly tighter than the standard lower bound calculation in

		TS	ACO		LBBD 2%		B&C Δ values (Δ′)				
|N|	|M|	Δ	Δ	Time (s)	Δ	Time (s)	30s	60s	600s	1800s	10800s
40	2	6.45	2.15	127.2	2.47	6.23	2.21 (0.03)	**2.00** (0.00)	**2.00** (0.00)	**2.00** (0.00)	**2.00** (0.00)
60	2	6.45	1.57	255.00	2.40	21.06	1.51 (0.22)	**1.40** (0.03)	**1.40** (0.00)	**1.40** (0.00)	**1.40** (0.00)
	4	8.17	4.54	192.60	**4.13**	240.13	8.74 (9.87)	**4.33** (3.84)	**3.69** (1.02)	**3.55** (0.83)	**3.18** (0.34)
80	2	5.95	1.25	416.40	1.72	58.11	2.02 (1.08)	1.70 (0.75)	**1.05** (0.01)	**1.05** (0.00)	**1.05** (0.00)
	4	7.66	3.97	311.40	**3.66**	749.44	60.07 (56.76)	5.59 (3.41)	**3.11** (0.94)	**2.94** (0.73)	**2.60** (0.34)
100	2	6.21	1.08	626.40	1.28	326.68	12.07 (11.83)	3.62 (3.41)	**0.84** (0.04)	**0.83** (0.02)	**0.81** (0.00)
	4	7.06	3.54	557.40	**3.24**	1080.14	*inf* (*inf*)	70.63 (64.17)	**2.91** (1.18)	**2.58** (0.80)	**2.34** (0.52)
	6	8.84	5.58	544.20	**4.61**	9716.32	*inf* (*inf*)	*inf* (*inf*)	**5.04** (2.84)	**4.65** (2.22)	**4.29** (1.84)
120	2	6.27	0.92	873.00	1.28	653.27	24.29 (23.58)	8.13 (6.24)	**0.65** (0.06)	**0.63** (0.03)	**0.61** (0.00)
	4	6.80	3.00	272.20	**2.98**	1142.06	*inf* (*inf*)	*inf* (*inf*)	**2.40** (1.15)	**2.20** (0.69)	**1.94** (0.41)
	6	8.20	4.52	753.00	**3.48**	10668.21	*inf* (*inf*)	*inf* (*inf*)	**4.03** (2.01)	**3.65** (1.60)	**3.20** (1.07)
	8	10.09	5.70	825.00	**4.56**	10800.00	*inf* (*inf*)	*inf* (*inf*)	**4.49** (2.04)	**4.47** (2.02)	**4.17** (1.63)

Table 2: Sub-optimal solution comparison. Solutions that are better than ACO have been bolded.

the metaheuristic literature. Not only does branch-and-check find better solutions than ACO in less time, it provides a much tighter estimate of the quality of that solution.

To provide a more detailed comparison between branch-and-check and ACO, Table 3 presents two views of the same data. The lines labelled with Δ correspond to the solution quality achieved by ACO and branch-and-check given the same average run-time. In the case of the branch-and-check results, we limited the run-time on each instance to the mean run-time of ACO for the corresponding problem size. For example, the column with $N = 40, M = 2$ indicates that ACO achieves a quality (mean relative error to LB) of 2.15% while branch-and-check achieves 2% in the same amount of time (127.2 seconds in this case). Conversely, the bottom two lines of Table 3 show the average run-time required to achieve the solution quality achieved by ACO. Again using the second column, ACO achieves its best solution quality in 127.2 seconds on average while branch-and-check achieves the same average quality in 5.4 seconds.

The Δ rows indicate that for every problem size, branch-and-check is able to provide better makespan solutions in the same amount of time. The run-time results show that the average time required to find solutions of equivalent quality is significantly shorter for branch-and-check than for ant colony optimization. The largest ratio is for the smallest problem (branch-and-check is about 23 times faster), but for the largest problem instance branch-and-check still achieves over a 2.5 times speed-up. Again, note that ACO and branch-and-check are not run on identical hardware so the CPU time comparison must be done with caution.

In addition to the strong heuristic performance of branch-and-check, since it is a complete search, it can prove optimality on many two-machine instances. Of the 75 instances we tested with two machines,

|N|	40	60		80		100			120			
|M|	2	2	4	2	4	2	4	6	2	4	6	8
Average quality given the same average run-time as ACO												
ACO (Δ)	2.15	1.57	4.54	1.25	3.97	1.08	3.54	5.58	0.92	3.00	4.52	5.70
B&C (Δ)	2.00	1.40	3.93	1.05	3.13	0.84	2.96	5.04	0.64	2.30	3.98	4.49
Average run-time to achieve the same quality as ACO												
ACO (s)	127.2	255.0	192.6	416.4	311.4	626.4	557.4	544.2	873.0	272.2	753.0	825.0
B&C (s)	5.4	17.3	136.0	53.3	64.0	73.3	233.9	245.1	97.3	260.0	577.6	297.6

Table 3: Performance comparison of ACO and branch-and-check. In the upper part of the table, the solution quality for ACO and branch-and-check at the time corresponding to the average solution time of ACO are presented. The lower part of the table presents the time required by each approach to achieve the average solution quality of ACO.

we were able to find and prove optimality for 64 of them within the three-hour time limit. Therefore, even though these instances have been considered for heuristic approaches, we are able to obtain globally optimal solutions in some cases.

6 Conclusion

We presented two decomposition approaches to minimize the makespan of an unrelated parallel machine scheduling problem with sequence and machine dependent setup times: logic-based Benders decomposition and branch-and-check. Both approaches decompose the problem in the same way, but differ in the timing of when the decomposed components are solved. 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. A TSP solver was then used to find optimal schedules for the jobs on each machine. We proved the convergence of these two decompositions and also showed how they could be used to find sub-optimal solutions.

The numerical results indicate that the cooperation of MIP and TSP solvers can effectively find optimal solutions. We are able to optimally solve instances six 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 four orders-of-magnitude faster. We showed that branch-and-check is consistently able to provide the best overall performance in terms of speed and solution quality. Furthermore, experiments illustrated that branch-and-check outperforms state-of-the-art metaheuristic approaches on the problem instances studied in the metaheuristic literature: the algorithm is able to find better solutions in less time and provide guarantees of solution quality.

This work illustrates the advantages of using decomposition methods, in particular branch-and-check, for the parallel machine scheduling problems with sequence and machine dependent setup times. Not only are these approaches able to find optimal solutions for much larger sized problems when compared to other exact methods, they are also able to outperform current metaheuristic approaches when problems are too large to find and prove optimal solutions.

References

[1] Achterberg, T. (2009). SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41.

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

[4] Aramon Bajestani, M. and Beck, J. (2013). Scheduling a dynamic aircraft repair shop with limited repair resources. *Journal of Artificial Intelligence Research*, 47:35–70.

[5] Arcus, A. (1965). A computer method of sequencing operations for assembly lines. *International Journal of Production Research*, 4(4):259–277.

[6] Arnaout, J., Rabadi, G., and Musa, R. (2010). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, 21(6):693–701.

[7] Avalos-Rosales, O., Alvarez, A., and Angel-Bello, F. (2013). A reformulation for the problem of scheduling unrelated parallel machines with sequence and machine dependent setup times. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS2013)*, pages 278–282.

[8] Baker, K. (1974). *Introduction to sequencing and scheduling.* John Wiley & Sons.

[9] Baldacci, R., Battarra, M., and Vigo, D. (2008). Routing a heterogeneous fleet of vehicles. *The vehicle routing problem: latest advances and new challenges*, pages 3–27.

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

[11] Benders, J. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252.

[12] Benini, L., Bertozzi, D., Guerri, A., and Milano, M. (2005). Allocation and scheduling for mpsocs via decomposition and no-good generation. In *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP2005)*, pages 107–121. Springer.

[13] 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*, pages 110–124. Springer.

[14] Fazel-Zarandi, M. and Beck, J. (2012). Using logic-based Benders decomposition to solve the capacity and distance constrained plant location problem. *INFORMS Journal on Computing*, 24:399–415.

[15] Fazel-Zarandi, M., Berman, O., and Beck, J. (2013). Solving a stochastic facility location/fleet management problem with logic-based Benders decomposition. *IIE Transactions*, 45(8):896–911.

[16] 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*, pages 92–101.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

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

[23] Hooker, J. (2000). *Logic-based Methods for Optimization.* Wiley.

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

[25] Hooker, J. (2007). Planning and scheduling by logic-based benders decomposition. *Operations Research*, 55(3):588.

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

[27] Karp, R. (1972). *Reducibility among combinatorial problems.* Springer.

[28] 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.

[29] Lee, J.-H., Yu, J.-M., and Lee, D.-H. (2013). A tabu search algorithm for unrelated parallel machine scheduling with sequence-and machine-dependent setups: minimizing total tardiness. *The International Journal of Advanced Manufacturing Technology*, pages 1–9.

[30] Lombardi, M. and Milano, M. (2012). Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. *Constraints*, 17(1):51–85.

[31] Rabadi, G. (2008). Scheduling research virtual center: http://schedulingresearch.com/.

[32] 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.

[33] Rocha, P., Ravetti, M., Mateus, G., and Pardalos, P. (2008). Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers & Operations Research*, 35(4):1250–1264.

[34] Terekhov, D., Beck, J., and Brown, K. (2009). A constraint programming approach for solving a queueing design and control problem. *INFORMS Journal on Computing*, 21(4):549–561.

[35] 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)*, pages 16–30. Springer.

[36] Tran, T. and Beck, J. (2012). Logic-based Benders Decomposition for Alternative Resource Scheduling with Sequence-Dependent Setups. In *Proceedings of the Twentieth European Conference on Artificial Intelligence (ECAI2012)*, pages 774–779.

[37] Valle, C., Martinez, L., Da Cunha, A., and Mateus, G. (2011). Heuristic and exact algorithms for a min–max selective vehicle routing problem. *Computers & Operations Research*, 38(7):1054–1065.