The two-machine flowshop total completion time problem: branch-and-bound algorithms based on network-flow formulation

Boris Detienne[a,b,*], Ruslan Sadykov[b,a], Shunji Tanaka[c]

[a] *Institute of Mathematics, University of Bordeaux, France*
[b] *Inria Bordeaux — Sud-Ouest, France*
[c] *Institute for Liberal Arts and Sciences, Department of Electrical Engineering, Kyoto University, Japan*

Abstract

We consider the flowshop problem on two machines with sequence-independent setup times to minimize total completion time. Large scale network flow formulations of the problem are suggested together with strong Lagrangian bounds based on these formulations. To cope with their size, filtering procedures are developed. To solve the problem to optimality, we embed the Lagrangian bounds into two branch-and-bound algorithms. The best algorithm is able to solve all 100-jobs instances of our testbed with and without setup times, thus significantly outperforming the best algorithms in the literature.

Keywords: Scheduling; Flowshop; Branch-and-bound; Lagrangean relaxation

1. Introduction

Problem description.. We consider the problem of scheduling a set of jobs $J = \{1, \ldots, n\}$ in a two-machine flowshop with the objective of minimizing the sum of completion times of jobs. The jobs are available at time zero and they should be processed first on machine 1, and then on machine 2. Each machine can process at most one job at a time. Let p_j^m denote the processing time of job j on machine m, where $m = 1, 2$. All processing times are integer. Preemption of the processing of the jobs in not allowed on either machine. Let C_j^m denote the completion time of job j on machine m. According to the scheduling classification, the problem is denoted by $F2||\sum C_j$. It is known to be NP-hard in the strong sense [11]. It has been shown by Conway et al. [7] that there exists at least one optimal solution where both machines have the same sequence of jobs. Thus, we may restrict the search to permutation schedules only.

In addition to this classic two-machine flowshop problem, we also consider its extension in which every job should be set up on each machine before being processed. Let s_j^m denote the setup time of job j on machine m. Setup of a job on machine 2 and processing of the same job on machine 1 can be performed in parallel. Note that setup times do not depend on the job processed just before job j, i.e. the setup times are *sequence independent*. This generalisation of the problem has been treated previously by Gharbi et al. [12]. It can be denoted as $F2|ST_{si}|\sum C_j$ in the scheduling classification. The set of permutation schedules remains dominant for this generalization as indicated in [5].

Literature review.. The problem $F2||\sum C_j$ has been studied in the literature for many years. First lower bounds and the branch-and-bound algorithms based on them were proposed by Ignall and Schrage [18], Ahmadi and Bagchi [2], and Della Croce et al. [9]. In these papers, instances with 10, 15, and 30 jobs, respectively, have been solved to optimality. Note that the processing times of jobs here are quite short and do not exceed 20 time units (even 10 in [9]).

Several Lagrangian relaxation-based lower bounds have been proposed for the problem. van de Velde [26] relaxed precedence constraints between operations of the same job to obtain a lower bound. The Lagrangian relaxation subproblem is difficult in the case where the schedule is restricted to be a permutation one. However, this subproblem becomes polynomially solvable when we further restrict Lagrangian

[*] Corresponding author. Phone: +33 (0)5.40.00.36.44
Email address: boris.detienne@math.u-bordeaux.fr (Boris Detienne)

multipliers to a same value. This restriction makes the Lagrangian bound weaker but computable in a short time. A perturbation procedure is used to improve this bound which consists in modifying the Lagrangian multipliers so as not to break the optimality of the current solution. Instances with up to 20 jobs have been solved to optimality using this bound. Hoogeveen and van de Velde [16] have improved this lower bound by adding slack variables to the precedence constraints and by transforming these constraints to equalities. However, this improved lower bounds has not been tested inside an enumeration algorithm. Della Croce et al. [8] used the same Lagrangian lower bound, but improved the perturbation procedure used by [26]. They also introduced some dominance relations to reduce the enumeration in the branch-and-bound algorithms. Instances with up to 45 jobs with short processing times (up to 10) and up to 30 jobs with long processing times have been solved to optimality.

A positional (assignment) formulation for the problem $F2||\sum C_j$ has been proposed independently by Akkan and Karabati [3] and Hoogeveen et al. [15]. In both works, the authors use the notion of waiting time of a job before its processing starts on the second machine. The formulation has $O(n^2)$ variables and $O(n)$ constraints. In [15], it was shown experimentally that the lower bound one can obtain by solving the linear relaxation of the positional formulation is stronger than any other bound proposed previously in the literature. It was also shown that any Lagrangian relaxation does not improve this bound. In [3], a network flow formulation for the problem was also suggested. In this network, each node corresponds to a position in the schedule and the waiting time of the job on this position. The network was then reduced by finding bounds on waiting times of jobs on different positions. To find a lower bound and design a branch-and-bound, the Lagrangian relaxation is used, in which job occurrence constraints are relaxed. Here the subproblem is the shortest path problem, and the Lagrangian dual problem is solved using a subgradient method. The branch-and-bound algorithm which uses this Lagrangian relaxation is able to solve instances with up to 60 jobs with small processing times (up to 10) and up to 45 jobs with large processing times (up to 100).

Haouari and Kharbeche [13] proposed valid inequalities for the positional formulation. They experimentally showed that the dual bound of the linear relaxation of the positional formulation is improved when these inequalities are added. However, these improved dual bounds were not embedded in any exact algorithm for the solution of the problem.

Hoogeveen and Kawaguchi [14] considered several special cases of the problem $F2||\sum C_j$. They proposed approximation algorithms for the general case of the problem as well as for a special case which they proved to be NP-hard. Three special cases of the problem were proved to be polynomially solvable.

There were two most important recent contributions to the problem $F2|ST_{si}|\sum C_j$ with sequence-independent setup times. Allahverdi [4] proposed two dominance relations and a branch-and-bound algorithm based on them. With this algorithm, all instances with up to 20 jobs with large processing and setup times (up to 100) were solved to optimality.

Gharbi et al. [12] proposed several dual bounds for the problem $F2|ST_{si}|\sum C_j$. Some of the suggested lower bounding procedures are similar to those used for the problem without setup times. One lower bound is based on solving the linear relaxation of a positional formulation. Another lower bound is based on Lagrangian relaxation similar to one used in [26]. Best exact algorithms based on the proposed dual bounds allowed the authors to solve all instances with up to 30 jobs and the majority of instances with 35 jobs with large processing and setup times (up to 100).

Our contribution.. In this work, we propose improved branch-and-bound algorithms for the problem $F2||\sum C_j$ as well as for its extension $F2|ST_{si}|\sum C_j$. Our approach is based on the network flow formulation from [3]. To obtain stronger dual bounds, we use a larger network than the one used in [3]. Different dominance rules and filtering techniques are exploited in order to cope with the size of the network. The structure of the network allows us to compute an expensive Lagrangian dual bound only once at the root node, and then recompute the bound in linear time at every node of the enumeration tree. Thus, millions of nodes can be checked in a reasonable time. Using the proposed algorithm, we are able to solve all instances of both problems $F2||\sum C_j$ and $F2|ST_{si}|\sum C_j$ with up to 100 jobs with large processing times.

The outline of the paper is as follows. In Section 2, we give the classic assignment MIP formulation of the problem. Different dominance rules, which are both from the literature and new ones, are described in Section 3. In Section 4, we present network flow formulations for the problem, as well as a subgradient algorithm with embedded filtering procedures to obtain Lagrangian dual bounds. Two improved branch-and-bound algorithms for the problem are suggested in Section 5. Results of computational experiments with these algorithms are given in Section 6. Finally, in Section 7, conclusions are drawn.

2. Mixed-integer linear programming formulation

This section introduces a mixed-integer linear programming formulation for the problem $F2|ST_{si}|\sum C_j$, which generalizes the positional formulation proposed in [3].

Note that the setup time of any job on machine 1 can be integrated into its processing time on machine 1. This follows from the fact that there exists an optimal schedule in which machine 1 process jobs without idle time. So, without loss of generality, for all $j \in J$, we can set $s_j^1 = 0$, and adjust appropriately processing times p_j^1.

In the following, $[k]$ denotes the index of the job in position k. The completion times $C_{[k]}^m$ of the job in position k, $k \in J$, on machines $m = 1, 2$ can be computed as:

$$\begin{aligned} C_{[k]}^1 &= C_{[k-1]}^1 + p_{[k]}^1. & (1)\\ C_{[k]}^2 &= \max\{C_{[k]}^1, C_{[k-1]}^2 + s_{[k]}^2\} + p_{[k]}^2. & (2) \end{aligned}$$

In [3], the authors introduced the notion of time lag between the processing of the same job on both machines to write a positional model and a network flow model for the problem. This kind of models is also called *waiting time-based* [models] in [12].

The completion-to-completion lag L_k^c of the job in position k, $k \in J$ is defined as the time elapsed between the completion of the job on machine 1 and its completion on machine 2:

$$\begin{aligned} L_k^c &= C_{[k]}^2 - C_{[k]}^1 \\ &= \max\{0, L_{k-1}^c + s_{[k]}^2 - p_{[k]}^1\} + p_{[k]}^2. \end{aligned}$$

The completion-to-start lag L_k^s of the job in position k, $k \in J$, is defined as the time elapsed between the completion of the job on machine 1 and its start on machine 2:

$$L_k^s = L_k^c - p_{[k]}^2 = \max\{0, L_{k-1}^s + p_{[k-1]}^2 + s_{[k]}^2 - p_{[k]}^1\}.$$

In order to write a convenient MILP model, the objective function is rewritten as:

$$\begin{aligned} \sum_k C_{[k]}^2 &= \sum_k (C_{[k]}^1 + L_k^c) \\ &= \sum_k \left((n-k+1)p_{[k]}^1 + L_k^s + p_{[k]}^2\right). \end{aligned}$$

Let a binary variable x_{jk}, $j, k \in J$, determine whether job j is processed in position k in the schedule. Let a continuous variable L_k^s, $k \in J$, represent the completion-to-start lag of the job in position k in the schedule. We now write the positional formulation for the problem $F2|ST_{si}|\sum C_j$.

$$\begin{aligned} \min \quad & \sum_{k=1}^n (n-k+1)\left(\sum_{j=1}^n p_j^1 x_{jk}\right) + \sum_{k=1}^n L_k^s + \sum_{j=1}^n p_j^2, & (3)\\ s.t. \quad L_1^s &\geq \sum_{j=1}^n (s_j^2 - p_j^1)x_{j1}, & (4)\\ L_k^s &\geq L_{k-1}^s + \sum_{j=1}^n p_j^2 x_{j,k-1} + \sum_{j=1}^n (s_j^2 - p_j^1)x_{jk}, \quad k = 2,\ldots,n, & (5)\\ \sum_{j=1}^n x_{jk} &= 1, \quad k = 1,\ldots,n, & (6)\\ \sum_{k=1}^n x_{jk} &= 1, \quad j = 1,\ldots,n, & (7)\\ L_k^s &\geq 0, \quad k = 1,\ldots,n, & (8)\\ x_{jk} &\in \{0,1\}, \quad j = 1,\ldots,n, k = 1,\ldots,n. & (9) \end{aligned}$$

Table 1 reports results obtained using Cplex 12.6 on a laptop with 16 GB RAM and an Intel i7 2.7 GHz processor. In order to keep the analysis concise, we only report in this section results for the problem without setup times. The test bed is composed of 20 instances (without setup times) for each combination (# of jobs, duration range) (see section 6 for details about the instances). The scalability of this straightforward approach is clearly limited.

Durations	n=10	n=30	n=40	n=50	n=60
$[1, 10]$	20	20	20	8	2
$[1, 100]$	20	20	8	0	1

Table 1: Number of instances of $F_2||\sum C_i$ solved to optimality within 1000 seconds using the model (3)-(9).

3. Dominance rules

This section introduces dominance rules which allow us to speed up the solution process. As specified in the corresponding sections, they are used in the branch-and-bound procedures to avoid exploring the set of solutions which are proved to be dominated, and in network reduction procedures to shrink the size of the networks.

The next proposition gives a generalization of the dominance rule from [26] for the problem $F2||\sum C_j$.

Proposition 1 ([4]). *If jobs i and j satisfy $p_i^1 + s_j^2 \leq p_j^1 + s_i^2$, $p_i^2 + s_i^2 \leq p_j^2 + s_j^2$, and $p_j^2 \leq p_i^2$, then there exists an optimal schedule in which job i precedes job j.*

In our solution methods, Proposition 1 is used as preprocessing to determine a set of predecessors Γ_i^- and successors Γ_i^+ for each job i. The next proposition is an extension of the dominance rule of [9] to the case with sequence-independent setup times.

Proposition 2. *Let σ be a partial schedule, and i and j two jobs not in σ. If $p_i^1 \leq p_j^1$, $p_i^2 \geq p_j^2$, $s_i^2 \geq s_j^2$, and*

$$\max\{C_\sigma^1 + p_i^1, C_\sigma^2 + s_i^2\} + p_i^2 \leq \max\{C_\sigma^1 + p_j^1, C_\sigma^2 + s_j^2\} + p_j^2, \tag{10}$$

then there exists an optimal solution not headed by σj, where σj denotes σ immediately followed by j.

Proof. Suppose there exists an optimal schedule S headed by σj. Let K be the set of positions of jobs scheduled between jobs j and i in S. Let A be the set of positions of all jobs scheduled after job i (including job i itself) in S. We denote by $C(S)_{[k]}^m$ the completion time of the job in position k on machine m in schedule S.

We now interchange jobs j and i in S to obtain another schedule S'. Then, from $p_i^1 \leq p_j^1$, we have

$$C(S')_{[k]}^1 \leq C(S)_{[k]}^1, \quad \forall k \in J. \tag{11}$$

Now, from (10) and (11) we can conclude that

$$C(S')_{[k]}^2 \leq C(S)_{[k]}^2, \quad \forall k \in K. \tag{12}$$

Finally, from (12), $p_i^2 \geq p_j^2$, and $s_i^2 \geq s_j^2$ it follows that $C(S')_{[k]}^2 \leq C(S)_{[k]}^2$, for all $k \in A$. □

Proposition 2 can be translated in terms of completion-to-completion lag, by substituting $L_\sigma^c = C_\sigma^2 - C_\sigma^1$:

Proposition 3. *Let σ be a partial schedule, L_σ^c the corresponding completion-to-completion lag, and i and j two jobs not in σ. If $p_i^1 \leq p_j^1$, $p_i^2 \geq p_j^2$, $s_i^2 \geq s_j^2$, and $\max(p_i^1, L_\sigma^c + s_i^2) + p_i^2 \leq \max(p_j^1, L_\sigma^c + s_j^2) + p_j^2$, then there exists an optimal solution not headed by σj.*

Let us consider a partial schedule, built up to position $k-1$. For a position k, two jobs a and b and $l \in \{k, k+1\}$, let $L_l^c(k, a \to b)$ denote the time lag between the completion times of $[l]$ on machines 1 and 2, under the assumption that a is processed at position k and b at position $k+1$ (i.e. $[k] = a$ and $[k+1] = b$):

$$L_k^c(k, a \to b) = \max\{p_a^1, L_{k-1}^c + s_a^2\} + p_a^2 - p_a^1 \quad (13)$$

$$L_{k+1}^c(k, a \to b) = \max\{p_b^1, L_k^c + s_b^2\} + p_b^2 - p_b^1 \quad (14)$$

Let us define $f(k, a \to b)$ as the cost generated by jobs a and b scheduled at positions k and $k+1$ respectively:

$$f(k, a \to b) = (n-k+1)p_a^1 + L_k^c(k, a \to b) + (n-k)p_b^1 + L_{k+1}^c(k, a \to b) \quad (15)$$

In the same way, one can define $L_l^c(k, \sigma)$ and $f(k, \sigma)$, which are the completion-to-completion lag in position l and the partial cost, respectively, when scheduling a sequence of jobs σ starting at position k.

The next proposition may be useful to improve a bound and/or to reduce the size of networks which will be presented below. It shows that, under some conditions, schedules in which a job j at position k precedes a job i can be discarded.

Proposition 4 (Dominance on two consecutive jobs). *If $f(k, i \to j) < f(k, j \to i)$ and $L_{k+1}^c(k, i \to j) \leq L_{k+1}^c(k, j \to i)$, then jobs j and i are not processed at positions k and $k+1$, respectively, in any optimal solution.*

Proof. $f(k, i \to j) < f(k, j \to i)$ means that the partial schedule up to position $k+1$ is more costly when jobs j and i are processed in positions k and $k+1$, respectively, than when their positions are interchanged.

In the partial schedules, the completion time of the job in position $k+1$ on machine 1 is $C_{[k-1]}^1 + p_i^1 + p_j^1$ regardless of the processing order of jobs i and j. It follows that $L_{k+1}^c(k, i \to j) \leq L_{k+1}^c(k, j \to i)$ implies $C_{[k+1]}^2(k, i \to j) \leq C_{[k+1]}^2(k, j \to i)$. As the cost function is non-decreasing, the partial schedule starting at position $k+2$ will not be more costly than when job j is processed before job i. □

The next proposition is utilized in our branch-and-bound procedures, as well as for removing edges in the networks (see Section 4). In the latter context, the constraint specifying that each job must be scheduled not more than once is relaxed. Hence, Proposition 5 is described in such a form that job repetition is allowed in partial sequences.

Proposition 5 (Dominance on K consecutive jobs). *Let $\sigma = (\sigma_1, \ldots, \sigma_l)$ be a partial sequence of jobs starting at position k, and let σ' be a permutation of σ. If at least one of these conditions hold, then no optimal schedule contains partial sequence σ of jobs starting at position k:*

- *$\sigma_i = \sigma_j$ for some $i \neq j$.*
- *$\sigma_j \in \Gamma_{\sigma_i}^-$ for some $i < j$.*
- *$f(k, \sigma') < f(k, \sigma)$ and $L_{k+|\sigma|-1}^c(k, \sigma') \leq L_{k+|\sigma|-1}^c(k, \sigma)$.*

Proof. Similar to the proof of Proposition 4. □

Consistency of the different rules. We should be careful in applying several dominance rules at the same time so that their consistency is ensured. For example, it can happen that one rule eliminates such schedules where job i precedes job j, whereas another rule forbids job j to precede job i. To ensure that at least one optimal schedule is not eliminated, we use a modified version of Proposition 3 where the condition $\max(p_i^1, L_\sigma^c + s_i^2) + p_i^2 \leq \max(p_j^1, L_\sigma^c + s_j^2) + p_j^2$ is replaced by $\max(p_i^1, L_\sigma^c + s_i^2) + p_i^2 < \max(p_j^1, L_\sigma^c + s_j^2) + p_j^2$. That way, any deduction made by Propositions 3, 4 or 5 removes only non-optimal schedules. Thus, when applying Proposition 1, we can safely break ties according to the index of the jobs.

4. Network flow formulations and lower bounds

In this section, we introduce two minimum cost flow formulations to obtain tight lower bounds. The first one is very similar to the one proposed in [3]. In order to get stronger bounds and solve larger instances, we also design an extended network.

4.1 Basic network G_1

Our first lower bound is based on a transshipment type network, which is a directed graph $G_1 = (V_1, A_1)$ whose structure is identical to the one proposed in [3]:

- Each node $v_{k,l} \in V_1$ of the network is associated with a position k in the sequence, and a value l of the completion-to-completion lag of the job in position $k-1$. Node $v_{1,0}$ is the source of the network. A sink node $v_{n+1,0}$ is also added, which represents the end of the schedule.
- For each combination of job j, position k, and completion-to-completion lag l, there is an arc $(v_{k,l}, v_{k+1,l'}, j) \in A_1$ from node $v_{k,l}$ to node $v_{k+1,l'}$. Here $l' = \max\{0, l + s_j^2 - p_j^1\} + p_j^2$ if $k < n$, and $l' = 0$ if $k = n$. This arc represents the processing of job j in position k, when the completion-to-completion lag of the previous job is equal to l, so that job j ends with a completion-to-completion lag equal to l'. Note that multiple arcs representing different jobs may connect the same pair of nodes. Following the expression of the objective function given by (3), the cost $c(v_{k,l}, v_{k+1,l'}, j)$ of using this arc is $(n-k+1)p_j^1 + l'$ when $k < n$, and $c(v_{n,l}, v_{n+1,0}, j) = p_j^1 + \max\{0, l + s_j^2 - p_j^1\} + p_j^2$.

Basic network flow formulation. The scheduling problem can be seen as the problem of finding a minimum cost flow of value 1 (a path) from the source node to the sink node, going through exactly one arc associated with each job, leading to the following ILP model:

$$\min \sum_{(v,w,j)\in A_1} c(v,w,j) \cdot x_{v,w,j} \tag{16}$$

$$s.t. \sum_{(v,w,j)\in A_1} x_{v,w,j} = \sum_{(w,v,j)\in A_1} x_{w,v,j} \qquad \forall v \in V_1 - \{v_{1,0}, v_{n+1,0}\} \tag{17}$$

$$\sum_{(v,w,j)\in A_1} x_{v,w,j} \leq 1 \qquad \forall j = 1, \ldots, n \tag{18}$$

$$\sum_{(v_{1,0},w,j)\in A_1} x_{v_{1,0},w,j} = 1 \tag{19}$$

$$x_{v,w,j} \in \{0,1\} \qquad \forall (v,w,j) \in A_1 \tag{20}$$

Network reduction during its creation. In order to reduce the size of the network, we use a procedure similar to the one described in [3]:

- An upper bound $\bar{z}$ on the optimum value of the problem is computed. Instead of the meta-heuristic of [9], we use the iterated enhanced dynasearch heuristic of [23], with a straightforward adaptation to handle setup times. The time taken by this heuristic is given in Table 2.
- A modified version of the positional model (3)-(9) is used to derive upper bounds on the lag values at each position, in all schedules whose cost is not larger than $\bar{z}$. Bounds on completion-to-completion lags allow one to remove dominated nodes. Bounds on completion-to-start lags and on the sum of those for consecutive pairs of positions allow one to remove dominated arcs. See [3] for details. In our implementation, the bounds are improved by removing variables x_{jk} from the model when job j cannot be processed in position k from the dominance rules, that is, $k \leq |\Gamma_j^-|$ or $k \geq n - |\Gamma_j^+|$.
- For each node v, we keep track of the set of jobs $\mathcal{P}(v)$ that exist in all paths from the source node to v, and utilize it when creating arcs out of v as follows:
  - Similar to [3], if $j \in \mathcal{P}(v)$, then no arc is created for job j out of v.
  - If $i \in \mathcal{P}(v)$ and $j \in \Gamma_i^-$, then no arc is created for j out of v.
- For each node v, we keep track of the set of jobs $\bar{\mathcal{P}}(v)$ that exist in at least one path from the source to v, and utilize it when creating arcs out of v as follows:
  - If $i \notin \bar{\mathcal{P}}(v)$ and $i \in \Gamma_j^-$, then no arc is created for j out of v.
  - If $i \notin \bar{\mathcal{P}}(v_{k,l})$ and j is dominated by i at lag l and position k in the sense of Proposition 3, then no arc is created for j out of v.

Table 2: Time (in seconds) required for computing the initial upper bound using heuristic from [23].

	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$ (instances of [12])			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	3.8	13.2	29.1	59.5				
$[1-100]$	5.2	16.6	38.1	80.0	27.5	43.4	65.4	129.4

As a preliminary experiment, we directly apply an MILP solver to formulation (16)–(20) based on network G_1 which is reduced as described just above. Table 3 reports the number of solved instances (without setup times) within the time limit. Table 6 reports the average number of nodes and arcs in graph G_1. It can be seen from the results that the network flow formulation is not much better than the positional formulation from Section 2.

Durations	n=10	n=30	n=40	n=50	n=60
$[1, 10]$	20	20	20	19	18
$[1, 100]$	20	15	2	0	0

Table 3: Number of instances of $F_2\|\|\sum C_i$ solved to optimality within 1000 seconds using formulation (16)-(20) based on G_1.

Lagrangian lower bound. The authors of [3] computed a Lagrangian lower bound based on a similar network. It is used within a branch-and-bound procedure, as well as some dominance rules to solve the problem. Relaxing the constraints (18) to the objective function leads to the following Lagrangian subproblem:

$$L_1(\pi) = \min \left\{ \sum_{(v,w,j)\in A_1} (c(v,w,j) + \pi_j)\, x_{v,w,j} - \sum_{j=1}^{n} \pi_j \middle| (17), (19), (20) \right\} \tag{21}$$

For any non-negative vector of Lagrange multipliers π, $L_1(\pi)$ is a Lagrangian lower bound on the optimum of the problem, which can be computed by solving a simple shortest path problem in G_1 with modified costs $c(v, w, j) + \pi_j$. This problem can be solved in $O(|A_1|)$-time complexity.

To improve bound $L_1(\pi)$, the constraint forbidding immediate repetition of a same job is added to the Lagrangian subproblem. We can take into account this constraint without increasing the complexity of the shortest path algorithm [1, 20].

Like in [24], we obtain near-optimal Lagrangian multipliers employing the conjugate subgradient algorithm [22, 21]. At each iteration k of the procedure, given current Lagrangian multipliers π^k, we first compute $L_1(\pi^k)$. We denote by x^{*k} the corresponding optimal solution of the Lagrangian subproblem. We choose the subgradient vector as $(g^k_j = 1 - \sum_{(v,w,j)\in A_1} x^{*k}_{v,w,j})_{j\in\{1,\ldots,n\}}$. The Lagrangian multipliers are updated by

$$d^k = g^k + \xi^k d^{k-1}$$
$$\pi^{k+1} = \pi^k + \gamma^k \frac{UB - L_1(\pi^k)}{||d^k||^2} d^k$$

Following [22, 21], we choose $\xi^k = ||g^k||/||d^{k-1}||$. The choice of the step size parameter γ^k obeys these rules:

- The initial value is $\gamma^0 = \gamma^{ini}$.
- It is decreased by $\xi^k = \kappa_S \gamma^{k-1}$ if the best lower bound is not updated for δ_S successive iterations.
- It is increased by $\xi^k = \kappa_E \gamma^{k-1}$ if the best lower bound is improved at iteration k.

The algorithm is terminated if the best lower bound does not increase by $100\epsilon/(1-\epsilon)\%$ and the gap between the best lower and upper bounds does not decrease by $100\epsilon\%$ in δ_T successive iterations, but not before min_{iter} iterations have been completed.

4.2 Filtering procedure for G_1

In order to reduce further the size of network G_1, we developed Lagrangian cost variable fixing [17, 24]. Given a vector of Lagrange multipliers π,

- let $F^1(v, \pi)$ be the cost of the shortest path from the source to node v in G_1 with modified costs;
- let $B^1(v, \pi)$ be the cost of the shortest path from node v to the sink in G_1 with modified costs.

Given π, values $F^1(v, \pi)$ for all nodes $v \in V_1$ can be found using the forward dynamic programming algorithm, and all values $B^1(v, \pi)$, $v \in V_1$ can be found using backward dynamic programming algorithm. Then an arc $(v, w, j) \in A_1$ can be removed from G_1 if $F^1(v, \pi) + c(v, w, j) + \pi_j + B^1(w, \pi) \geq \bar{z}$. Furthermore, a node $v \in V_1$ can be removed from G_1 if it does not have any incoming or any outgoing arc anymore. For a fixed vector π, the time complexity of this filtering procedure remains equal to $O(|A_1|)$. The filtering procedure is called at every iteration of the conjugate subgradient algorithm.

The running time of the subgradient algorithm is presented in Table 4. The relative gaps obtained after the subgradient algorithm are shown in Table 5. The gap is computed as $\frac{UB-LB}{LB}$, where UB is the upper bound given by the heuristic [23], and LB is the best Lagrangian bound obtained during the subgradient algorithm. One can see from the results that a very tight lower bound is obtained in small running time, which is 25 seconds for the largest instances without setup times. The computing time is twice larger for instances with setup times. This is explained by the slightly degraded performance of the dynasearch procedure for this class of problems to compute an upper bound (the root gap is twice larger). Hence, the network cannot be reduced as much in this case (as can be seen from Table 6), so that solving the subproblem at each iteration of the subgradient procedure takes more time.

Table 4: Average running time (in seconds) for the subgradient procedure on network G_1.

	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.2	0.4	0.8	1.7				
$[1-100]$	1.1	4.2	10.5	23.5	9.0	16.1	24.7	55.9

Table 5: Average duality gap produced by the subgradient procedure on network G_1.

	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.13%	0.11%	0.10%	0.08%				
$[1-100]$	0.16%	0.18%	0.12%	0.10%	0.21 %	0.22 %	0.20 %	0.20 %

From Table 6 it can be seen that the filtering procedure reduces the size of the graph G_1 significantly: the number of arcs is decreased by a factor of 5 on large instances without setup times, and by more than 2 on instances with setup times. Once again, the degraded results can be attributed to the quality of the upper bound used in the filtering procedure.

We applied an MILP solver to formulation (16)–(20) based on filtered network G_1'. Table 7 reports the number of solved instances (without setup times) within the time limit. Again the improvement of results in comparison with the network flow formulation based on G_1 is very limited despite a significant reduction of the graph size.

4.3 Expanded network G_2

Although lower bounds obtained using Lagrangian relaxation of the network flow formulation based on G_1 are already very tight, we can improve them by adding another dimension to G_1. It results in a larger network G_2, which allows us to eliminate more dominated subsequences of jobs in the Lagrangian subproblem and thus to improve the lower bound.

Table 6: Average size of network G_1 before and after filtering.

Number of nodes in G_1, thousands								
	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.8	1.3	2.0	2.8				
$[1-100]$	8.0	14.6	21.3	30.4	20.3	26.1	31.9	46.3
Number of arcs in G_1, thousands								
	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	21.9	59.3	119.0	213.9				
$[1-100]$	258.3	731.7	1451.6	2635.2	1026.4	1553.3	2189.3	4030.1
Number of nodes in G_1 after filtering, thousands								
	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.5	0.9	1.4	1.9				
$[1-100]$	4.4	9.1	14.2	21.0	14.9	20.3	24.9	38.6
Number of arcs in G_1 after filtering, thousands								
	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	3.4	11.3	26.2	47.0				
$[1-100]$	30.9	129.8	264.0	520.6	311.6	567.6	828.6	1813.3

Table 7: Number of instances of $F2||\sum C_i$ solved to optimality within 1000 seconds using formulation (16)-(20) based on G'_1.

Durations	n=10	n=30	n=40	n=50	n=60
$[1,10]$	20	20	20	19	19
$[1,100]$	20	19	9	2	1

Network structure of G_2.. The network is a directed graph $G_2 = (V_2, A_2)$ with the following structure.

- Each node $v_{k,l,i} \in V_2$ of the network is associated with a position k in the sequence, a job i, and a value l of the completion-to-completion lag of the job in position $k-1$. Node $v_{0,0,0}$ is the source of the network. An additional sink node $v_{n+1,0,0}$ is added, which represents the end of the schedule.
- For each combination of jobs i, j, $i \neq j$, position k, and completion-to-completion lag l, there is an arc $(v_{k,l,i}, v_{k+1,l',j}) \in A_2$, with:

$$l' = \begin{cases} 0 & \text{if } k = 0 \\ \max\{0, l + s_i^2 - p_i^1\} + p_i^2 & \text{otherwise} \end{cases}$$

  This arc represents the processing of job i in position k, when the completion-to-completion lag of the job in position $k-1$ is equal to l, the completion-to-completion lag of job i is equal to l', and job j is processed at position $k+1$. When $0 < k < n$, the cost $c(v_{k,l,i}, v_{k+1,l',j})$ of using this arc is $(n-k+1)p_i^1 + l'$. For the first position, $c(v_{0,0,0}, v_{1,0,j}) = 0$. For the last position, $c(v_{n,l,i}, v_{n+1,0,0}) = p_i^1 + \max\{0, l + s_i^2 - p_i^1\} + p_i^2$.

Network reduction during its creation..

- A node $v_{k,l,i}$ in G_2 is created only if an arc $(v_{k,l}, v_{k+1,l'}, i)$ exists in filtered network G'_1.
- Arc $(v_{k,l,j}, v_{k+1,l',i})$ is not created if scheduling job j is dominated at lag l by Proposition 3.

- Arc $(v_{k,l,j}, v_{k+1,l',i})$ is not created if scheduling of jobs j and i at positions k and $k+1$, respectively, is dominated by Proposition 4.
- Arc $(v_{k,l_1,j_1}, v_{k+1,l_2,j_2})$ is not created if, for all arcs $(v_{k-1,l_0,j_0}, v_{k,l_1,j_1})$, the sequence of jobs (j_0, j_1, j_2) is dominated at lag l_0 and position $k-1$ according to Proposition 5.

4.4 Filtering procedure for G_2

We developed a filtering procedure embedded in the subgradient algorithm. The procedure is similar to the one described in Section 4.2, but has the following differences.

- The Lagrangian lower bound is improved using 3-cycle elimination by forbidding such partial sequences of job as (i, j, i) (adding this constraint does does change the time-complexity of the Lagrangian subproblem [1, 20]).
- Similar to the filtering procedure in Section 4.2, Lagrangian cost fixing is performed by computing the following values.
  - $F^2(v, \pi)$ is the cost of the shortest path from the source to node v in G_2 with modified costs;
  - $B^2(v, \pi)$ is the cost of the shortest path from node v to the sink in G_2 with modified costs.

  An arc $(v, w) \in A_2$ can be removed from G_2 if $F^2(v, \pi) + c(v, w) + \pi_j + B^2(w, \pi) \geq \bar{z}$, where j is the index of the job represented by w.
- In addition, Proposition 5 is applied to perform what we call 3-consecutive-jobs filtering. It removes arc $(v_{k,l_1,j_1}, v_{k+1,l_2,j_2})$ if one of these conditions holds:
  - for all arcs $(v_{k-1,l_0,j_0}, v_{k,l_1,j_1})$ in G_2, the sequence of jobs (j_0, j_1, j_2) is dominated at lag l_0 and position $k-1$ according to Proposition 5 ;
  - for all arcs $(v_{k+1,l_2,j_2}, v_{k+2,l_3,j_3})$ in G_2, the sequence of jobs (j_1, j_2, j_3) is dominated at lag l_1 and position k according to Proposition 5.

  This 3-consecutive-jobs filtering is costly. Therefore, it is not applied at every iteration of the subgradient procedure, but each time the number of arcs in the graph is reduced by 5% using the Lagrangian cost fixing since the last time 3-consecutive-jobs filtering was used.
- The lower bound is improved using Proposition 5 by removing dominated sequences of three jobs which are part of the Lagrangian subproblem solution. More precisely, at each iteration of the subgradient procedure, the Lagrangian subproblem solution is inspected. Assume that a subsequence of jobs (j_1, j_2, j_3) starting at lag l_1 and position k is part of the solution and dominated. Let $(v_{k,l_1,j_1}, v_{k+1,l_2,j_2}, v_{k+2,l_3,j_3})$ be the corresponding sequence of nodes. This path is removed from the network using the following procedure (see Figure 1):
  1. Identify the set $V^*(k, l_1, j_1, l_2, j_2)$ of nodes v' which are successors of v_{k+1,l_2,j_2} and such that sequence $(v_{k,l_1,j_1}, v_{k+1,l_2,j_2}, v')$ is not dominated (by inspection).
  2. Create a new node v'', which is a duplicate for node v_{k+1,l_2,j_2}.
  3. For each node $v' \in V^*(k, l_1, j_1, l_2, j_2)$, create an arc (v'', v'), which is a duplicate for (v_{k+1,l_2,j_2}, v').
  4. Create an arc (v_{k,l_1,j_1}, v'').
  5. Remove arc $(v_{k,l_1,j_1}, v_{k+1,l_2,j_2})$.

  The Lagrangian subproblem is then solved again, on the modified network and with the same Lagrange multipliers. This procedure is repeated until there exist no dominated sequences of three jobs in the Lagrangian subproblem solution. This solution is then returned to the sub-gradient procedure.
- At the end of the subgradient procedure, the following additional filtering procedure is used, as in [10]. Given a vector of Lagrange multipliers π,

- let $F^2_j(v,\pi)$ (resp. $B^2_j(v,\pi)$) denote the cost of the shortest path from the source to node v (resp. from node v to the sink) in G_2 with Lagrangian costs, such that this path goes through exactly one arc representing job j;
- let $F^2_{\neg j}(v,\pi)$ (resp. $B^2_{\neg j}(v,\pi)$) denote the cost of the shortest path from the source to node v (resp. from node v to the sink) in G_2 with Lagrangian costs, such that this path does not contain any arc representing job j.

Given π, these values for all jobs and all nodes can be computed in time $O(n|A_2|)$ by applying several times the forward and backward dynamic programming algorithms. Then for each arc $a = (v_{k,l,j}, v_{k+1,l',i}) \in A_2$, we compute value $L^2(a,\pi)$ as:

$$L^2(a,\pi) = \max \left\{ \begin{array}{l} F^2_{\neg j}(v_{k,l,j},\pi) + c(a) + \pi_j + B^2_{\neg j}(v_{k,l',i},\pi), \\ \max\limits_{j' \neq j} \left\{ \min \left\{ \begin{array}{l} F^2_{j'}(v_{k,l,j},\pi) + c(a) + \pi_j + B^2_{\neg j'}(v_{k,l',i},\pi), \\ F^2_{\neg j'}(v_{k,l,j},\pi) + c(a) + \pi_j + B^2_{j'}(v_{k,l',i},\pi) \end{array} \right\} \right\} \end{array} \right\}.$$

Arc a can be removed from G_2 if $L^2(a,\pi) > \bar{z}$.

The running time of the subgradient algorithm with embedded filtering of graph G_2 is presented in Table 8. The relative gaps obtained after the subgradient algorithm are shown in Table 9. One can see from the results that the gap is decreased by 30% in comparison with the subgradient procedure on network G_1. However, the running time is increased by a factor of 4.

Table 8: Time in seconds for the subgradient procedure on network G_2.

	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.2	0.7	2.5	8.9				
$[1-100]$	1.3	9.8	38.1	94.1	68.0	167.7	291.0	913.5

Table 9: The duality gap produced by the subgradient procedure on network G_2.

	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.07%	0.05%	0.06%	0.06%				
$[1-100]$	0.09%	0.07%	0.08%	0.07%	0.19%	0.20 %	0.19 %	0.19%

From Table 10 it can be seen that, again, the filtering procedure reduces the size of the graph significantly for instances without setup times: the number of arcs is decreased by a factor of 4 on large instances. For instances with setup times, the number of arcs is still very large. In Section 5, we show that filtering is also efficient for this class of instances when used with a tighter upper bound.

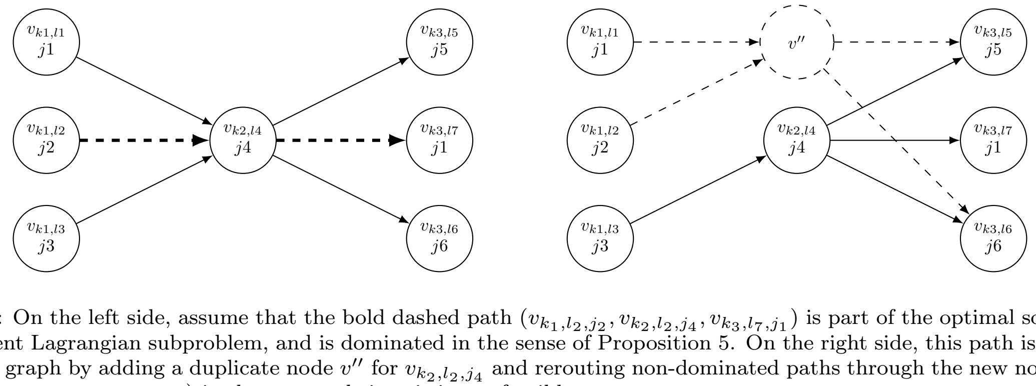

Figure 1: On the left side, assume that the bold dashed path $(v_{k_1,l_2,j_2}, v_{k_2,l_2,j_4}, v_{k_3,l_7,j_1})$ is part of the optimal solution of the current Lagrangian subproblem, and is dominated in the sense of Proposition 5. On the right side, this path is removed from the graph by adding a duplicate node v'' for v_{k_2,l_2,j_4} and rerouting non-dominated paths through the new node. Path $(v_{k_1,l_1,j_1}, v_{k_2,l_2,j_4}, v_{k_3,l_7,j_1})$ is also removed since it is not feasible.

Table 10: Average size of network G_2 before and after filtering. Filtering is performed with the initial upper bound.

Number of nodes in G_2, thousands

	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	2.3	7.8	17.0	35.8				
$[1-100]$	26.5	92.7	212.4	391.3	246.7	426.9	608.4	1 234.1

Number of arcs in G_2, thousands

	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	12.9	68.2	217.6	642.7				
$[1-100]$	164.2	937.0	2925.4	6431.4	3818.3	8224.6	13 550.5	35 554.8

Number of nodes in G_2 after filtering, thousands

	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.4	2.2	6.6	13.0				
$[1-100]$	5.2	35.5	92.5	166.2	163.7	284.8	396.8	766.3

Number of arcs in G_1 after filtering, thousands

	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.8	7.6	38.6	99.2				
$[1-100]$	16.4	170.7	639.0	1465.4	1866.5	4236.0	6931.7	18 544.7

5. Branch-and-bound algorithms

We have implemented two branch-and-bound algorithms.

- Algorithm BB_1 is based on network G_1 and Lagrangian bound $L^1(\pi)$.
- Algorithm BB_2 is based on network G_2 and Lagrangian bound $L^2(\pi)$.

The following parts of the algorithm are the same for both BB_1 and BB_2.

- The initial upper bound is computed by the dynasearch heuristic from [23].
- Network G_1 or G_2 is constructed and reduced by the corresponding filtering algorithm.
- After the subgradient algorithm, the vector of multipliers π^* which gives the best Lagrangian lower bound $L^1(\pi^*)$ or $L^2(\pi^*)$ is fixed till the branch-and-bound termination.
- The set of possible job sequences is explored, by enumerating the set of feasible (with respect to the job assignment constraint) paths in graph G_1 or G_2. We proceed from the source to the sink in the graph. For each node v, the outgoing arcs (v, w) are sorted in non-decreasing order of $B^1(w, \pi^*)$ or $B^2(w, \pi^*)$. The algorithms use the depth-first-search rule in this order.
- We use Proposition 5 in a Memory Dominance Rule fashion [6, 25, 19]: the set of non-dominated subsequences explored is maintained in a hash map. At each node of the tree, the current subsequence is tested against the subsequences composed of the same set of jobs.
- At each node of the search tree, we maintain incrementally the number of unscheduled predecessors of each job. An arc in G_1 or G_2 corresponding to job j is a candidate for branching only if the number of remaining predecessors of j is zero.
- We branch, i.e. we extend current partial sequence σ with job j, only if the subsequence of the last K jobs ($K = 5$ in our implementation) including j is not dominated, according to Proposition 5, by any permutation of these K jobs.

- The upper bound may be updated only when a leaf node of the search tree is reached. We do not use any heuristics within the branch-and-bound algorithm.

5.1 Algorithm BB_1

In algorithm BB_1, at each non-root node of the search tree corresponding to node v in graph G_1, we compute lower bound $L^1(v)$. Let σ be the partial sequence built so far. Then,

$$L^1(v) = cost(\sigma) + B^1(v, \pi^*) - \sum_{j \notin \sigma} \pi_j^*. \tag{22}$$

Computation of $L^1(v)$ can be done in constant time by incrementally maintaining at each node the current value of $\sum_{j \notin \sigma} \pi_j$.

Note that algorithm BB_1 is similar to the branch-and-bound algorithm presented in [3]. The main differences are that graph G_1 is filtered, and we use Proposition 5 as a memory dominance rule. In [3], other dominance rules are used, but preliminary computational experiments indicated that they do not improve the performance of our branch-and-bound procedure.

5.2 Algorithm BB_2

In algorithm BB_2, at each non-root node of the search tree corresponding to node v in graph G_2, we compute lower bound $L^2(v)$. Again, let σ be the partial sequence built so far. Then,

$$L^2(v) = cost(\sigma) + \max \left\{ \begin{array}{l} B^2(v, \pi^*), \\ \max_{j \notin \sigma} B_j^2(v, \pi^*), \\ \max_{j \in \sigma} B_{\neg j}^2(v, \pi^*) \end{array} \right\} - \sum_{j \notin \sigma} \pi_j^*. \tag{23}$$

Computation of $L^2(v)$ can be done in time $O(n)$.

Tentative upper bound. As one can expect from Tables 9 and 10, solving large instances of the problem $F2|ST_{SI}|\sum C_i$ by enumerating the set of paths in network G_2 is very time-consuming. The optimal objective values of medium size instances indicate us that the very large size of the network in this case compared with the case without setup times clearly comes from a degraded initial upper bound. That is why, following the idea described in [24], we introduce the use of a tentative upper bound around algorithm BB_2. The overall procedure can be summarize like this:

- Build and filter network G_1, to obtain network G_1'.
- If the number of arcs in G_1' does not exceed a given threshold (fixed empirically to 300 thousands arcs in our experiments), then build network G_2 from G_1', filter G_2 and run algorithm BB_2 to solve the problem to optimality.
- Otherwise, it is very likely that the upper bound significantly over-estimates the optimal objective value and that it will result in a very large network G_2 at the basis of the branch-and-bound procedure. In this case, perform a major iteration: use a tentative upper bound UB^{tent} to build and filter network G_2 from G_1', and run algorithm BB_2. Two outcomes are possible.
  - If algorithm BB_2 does not improve the upper bound, then UB^{tent} under-estimates the optimal objective value. In this case, the algorithm performs next major iteration with an increased value of UB^{tent}.
  - If algorithm BB_2 completes with a new, improved upper bound, then it is optimal and the overall procedure terminates.

Table 11 shows the usefulness of the tentative upper bound on instances with setup times: the size of network G_2 once it is filtered during the last major iteration (i.e. with the smallest feasible tentative upper bound) is more than seven times smaller than when it is filtered with the initial upper bound for 100-job instances. We do not report the corresponding results for instances without setup times since the impact is very limited for this class of problems.

Number of nodes in G_2 after filtering, thousands							
Initial upper bound				Best feasible tentative upper bound			
n=60	n=70	n=80	n=100	n=60	n=70	n=80	n=100
163.7	284.8	396.8	766.3	63.1	88.4	135.1	237.1
Number of arcs in G_2 after filtering, thousands							
Initial upper bound				Best feasible tentative upper bound			
n=60	n=70	n=80	n=100	n=60	n=70	n=80	n=100
1 866.5	4 236.0	6 931.7	18 544.7	344.1	544.5	1013.3	2 237.8

Table 11: Average size of network G_2 after filtering, for problem $F2|ST_{SI}|\sum C_i$, when filtering is performed with the initial upper bound and the smallest feasible tentative upper bound.

6. Computational results

All the algorithms were coded in C++ and compiled under Microsoft Visual Studio 2012. All the experiments were conducted on a laptop computer with an Intel i7 2.7 GHz processor and 16GB RAM. The solver used to solve the MILP and LP models is IBM ILOG Cplex v12.6.

The initial value of the tentative upper bound is chosen as $UB^{tent} = \alpha_1(UB - LB) + LB$, where UB and LB are, respectively, the initial upper bound obtained by the dynasearch procedure and the best lower bound at the end of the subgradient procedure applied to filter network G_1. If necessary, UB^{tent} is increased at each major iteration by $\alpha_2(UB - LB)$. In our implementation, $\alpha_1 = 0.4$ and $\alpha_2 = 0.2$, ensuring the convergence of the overall procedure in four major iterations.

We use the same tuning of the different parameters for subgradient procedure for filtering networks G_1 and G_2 for both problem types with and without setup times: $\gamma^{ini} = 1$, $\kappa_S = 0.95$, $\kappa_E = 1.02$, $\epsilon = 10^{-4}$, $\delta_S = 2$, $\delta_T = n$, $min_{iter} = 2n$.

6.1 Instances without setup times

We first tested our branch-and-bound algorithms on instances without setup times generated similarly to the instances in [13]. The instance generator takes as input the number of jobs n, and a maximum duration of operations p_{max}. The duration of the operations is drawn from the uniform distribution $[1, p_{max}]$. We generated 20 instances for each combination of parameters $n \in \{10, 30, 40, 50, 60, 70, 80, 90, 100\}$ and $p_{max} \in \{10, 100\}$.

In Table 12 we report the number of instances solved to optimality within the time limit of 1000 seconds (for the whole method) by both algorithms BB_1 and BB_2. Algorithm BB_2 solves all instances of the testbed within the time limit. The hardest instance is solved in 602 seconds.

Table 12: $F2||\sum C_i$: Number of instances of our testbed solved to optimality within 1000 seconds. The hardest instance is solved by BB_2 in 602 seconds.

Alg.	Duration	n=10	n=30	n=40	n=50	n=60	n=70	n=80	n=90	n=100
BB_1	$[1-10]$	20	20	20	20	20	20	20	18	18
BB_1	$[1-100]$	20	20	20	20	19	20	20	19	15
BB_2	$[1-10]$	20	20	20	20	20	20	20	20	20
BB_2	$[1-100]$	20	20	20	20	20	20	20	20	20

In Table 13 and Table 14 we give, respectively, the average running time of our branch-and-bound algorithms, and the average number of nodes in the search tree. These average values are computed on the instances solved to optimality by both methods.

As it can be seems from the results, algorithm BB_2 performs better than BB_1 in terms of computing time, and explores up to 70 times less nodes for instances with 100 jobs and large durations. The difference in the number of nodes can be explained by the fact that lower bound L^2 is much stronger than L^1 and reduces the size of the search tree by several orders of magnitude for some instances. Moreover, it allows an early detection of unfeasible subsequences that cannot be extended to complete sequences with each

Table 13: $F2||\sum C_i$: Average total running time (in seconds) of the algorithms. Time for initial heuristic, network building and filtering is included. For BB_2, the overall time including all major iterations of the tentative upper bound procedure is reported.

Alg.	Duration	n=10	n=30	n=40	n=50	n=60	n=70	n=80	n=90	n=100
BB_1	$[1-10]$	0.3	2.7	5.7	11.2	19.8	22.7	60.9	62.4	89.6
BB_1	$[1-100]$	0.2	3.4	8.7	16.9	33.3	60.8	113.6	339.1	455.6
BB_2	$[1-10]$	0.3	2.2	4.5	8.9	14.8	23.2	35.1	54.6	95.0
BB_2	$[1-100]$	0.3	3.0	8.2	17.4	34.2	57.3	91.8	153.6	215.8

Table 14: $F2||\sum C_i$: Average number of nodes for the branch-and-bound algorithms (K: thousands, M: millions). For BB_2, the total number of nodes explored in all major iterations of the tentative upper bound procedure is reported.

Alg.	Duration	n=10	n=30	n=40	n=50	n=60	n=70	n=80	n=90	n=100
BB_1	$[1-10]$	0.0 K	0.7 K	5.0 K	38.5 K	124.7 K	777.1 K	7.0 M	20.3 M	15.3 M
BB_1	$[1-100]$	0.0 K	1.7 K	78.0 K	320.6 K	2.6 M	23.3 M	53.0 M	214.0 M	283.5 M
BB_2	$[1-10]$	0.0 K	0.0 K	0.2 K	2.5 K	2.5 K	45.7 K	236.1 K	1.3 M	8.6 M
BB_2	$[1-100]$	0.0 K	0.0 K	0.0 K	4.0 K	12.2 K	303.4 K	348.2 K	3.1 M	3.9 M

job processed exactly once. For example, if job j is already scheduled, and no path without job j exists in G_2 from current node to the sink node, then $B_{\neg j}(v) = \infty$ and the branch-and-bound node is pruned by the bound. In less extreme scenarios, such paths exist but are all relatively costly, and $B_{\neg j}(v)$ may be sufficiently large to prune the node. The difference in the solution times is less. One can remark that the distribution of the time consumed by each algorithm is not the same: for BB_1, most of the time is spent in the exploration of the search tree, while for BB_2 the computational effort is balanced between the filtering of G_2 and the exploration of the search tree. The relatively small difference in solution times is also due to the fact that calculating L^2 takes linear time instead of constant time for bound L^1. Note that both algorithms BB_1 and BB_2 solved all the instances from the paper [13] (with up to 70 jobs) within 1000 second.

6.2 *Instances with setup times*

For the problem $F2|ST_{SI}|\sum C_i$, our solving methods are tested against the testbed of Gharbi et al. [12]. Their generator takes as input a number n of jobs, and a factor K for setup times. The processing time of each operation is drawn from the uniform distribution $[1, 100]$, and one setup time is drawn for each operation from the uniform distribution $[1, 100K]$. As mentioned in Section 2, we modify the instances by integrating the setup time of the first operation of each job into its processing time. The whole set of instances is composed of 50 instances for each combination of the number of jobs ranging from 10 to 500, and $K \in \{0.25, 0.5, 0.75, 1\}$. We restrict our computational study to the 800 instances with the number of jobs in $n \in \{60, 70, 80, 100\}$ (the test set of [12] does not contain 90-job instances).

Table 15 reports the number of instances solved to optimality by both methods within 1000 seconds. Algorithm BB_2 significantly outperforms BB_1. Moreover, algorithm BB_2 solves all instances of the testbed with up to 100 jobs in less than two hours: only four 100-job instances are not solved within one hour; the hardest instance is solved in 6443 seconds.

The value of parameter K has no significant impact on the performance of algorithm BB_1. However, while BB_2 solves 47 out of the 50 100-job instances within 1000 seconds when $K = 0.25$, it solves only 26 of the instances when $K = 1$ in the same time. This can be explained by the wider range of completion-to-completion lag in the later case, leading to more nodes in both networks G_1 and G_2. For large instances, the larger size of G_2 implies a consequent computational effort during the first iterations of the subgradient procedure (when the network is only a little shrunk). This hypothesis is confirmed by the fact that the root gap is not significantly impacted by parameter K, while the average time for the subgradient procedure is increased by a factor two for 80-job and 100-job instances when increasing parameter K from $K = 0.25$ to $K = 1$.

In Table 16 and Table 17 we give, respectively, the average running time of our branch-and-bound algorithms, and the average number of nodes in the search tree. These results emphasize the importance

Table 15: $F2|ST_{SI}|\sum C_i$: Number of instances of the testbed of [12] solved to optimality within 1000 seconds. The hardest instance is solved by algorithm BB_2 in 6443 seconds.

Alg.	n=60	n=70	n=80	n=100
BB_1	200	190	125	10
BB_2	200	200	200	145

of having tight lower and upper bounds in our methods: algorithm BB_2 performs two order of magnitude less nodes than algorithm BB_1.

Table 16: $F2|ST_{SI}|\sum C_i$: Average total running time (in seconds) of the algorithms. Time for initial heuristic, network building and filtering is included. For BB_2, the overall time including all major iterations of the tentative upper bound procedure is reported.

Alg.	n=60	n=70	n=80	n=100
BB_1 - Avg. on solved instances	79.6	280.3	393.9	615.8
BB_2 - Avg. on instances solved by BB_1	99.5	162.6	235.9	478.8
BB_2 - Avg. on all instances	99.5	168.6	287.18	935.2

Table 17: $F2|ST_{SI}|\sum C_i$: Average number of nodes for the branch-and-bound algorithms (K: thousands, M: millions). For BB_2, the total number of nodes explored in all major iterations of the tentative upper bound procedure is reported.

Alg.	n=60	n=70	n=80	n=100
BB_1 - Avg. on solved instances	30.4 M	142.6 M	216.2 M	301.1 M
BB_2 - Avg. on instances solved by BB_1	125.7 K	310.6 K	626.1 K	822.0 K
BB_2 - Avg. on all instances	125.7 K	369.2 K	2.4 M	42.6 M

7. Conclusions

In this paper, we proposed improved branch-and-bound algorithms for the flowshop problem on two machines with minimum flow time criterion, as well as for its extension with sequence-independent setup times. Our algorithms are based on large-scale network flow formulations of the problem. At the root node, a subgradient procedure is used to compute a very strong Lagrangian dual bound. To reduce the size of the network and improve the dual bound, a filtering technique (combined with the use of a tentative upper bound) and dominance rules are exploited (including a memory based dominance rule). Lagrange multipliers are not updated beyond the root node, which allows us to speed-up the calculation of the lower bound at non-root nodes of the search tree.

Our best algorithm is able to solve all tested 100-jobs instances coming from the literature or generated in the same way as in the literature. This is a significant improvement over the best known algorithm dedicated to the special case without setup times, which cannot solve instances with more than 60 jobs with small processing times and with more than 45 jobs with large processing times, as well as for the case with setup times for which the best published method cannot solve instances with more than 35 jobs.

Future research directions include the use of the Successive Sublimation Dynamic Programming [17, 24] algorithm to find an optimal feasible solution instead of branch-and-bound algorithms. It would also be interesting to investigate the possibility of applying similar approaches to more general problems (for example with general total cost functions or more machines) using adapted network representations of the problems.

Acknowledgements

The authors would like to thank Anis Gharbi, Mohamed Kais Msakni and their co-authors for providing us with their instances and detailed computational results, as well as Federico Della Croce and

Vincent T'Kindt for their useful discussion about the problem.

[1] T. S. Abdul-Razaq and C. N. Potts. Dynamic programming State-Space relaxation for Single-Machine scheduling. *The Journal of the Operational Research Society*, 39(2):141–152, 1988.

[2] Reza H. Ahmadi and Uttarayan Bagchi. Improved lower bounds for minimizing the sum of completion times of n jobs over m machines in a flow shop. *European Journal of Operational Research*, 44(3):331 – 336, 1990.

[3] Can Akkan and Selçuk Karabati. The two-machine flowshop total completion time problem: Improved lower bounds and a branch-and-bound algorithm. *European Journal of Operational Research*, 159(2):420–429, December 2004.

[4] Ali Allahverdi. Minimizing mean flowtime in a two-machine flowshop with sequence-independent setup times. *Computers and Operations Research*, 27(2):111 – 127, 2000.

[5] Ali Allahverdi, Jatinder N.D Gupta, and Tariq Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 27(2):219 – 239, 1999.

[6] Ph. Baptiste, J. Carlier, and A. Jouglet. A branch-and-bound procedure to minimize total tardiness on one machine with arbitrary release dates. *European Journal of Operational Research*, 158(3):595–608, 2004.

[7] R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling.* Addison-Wesley, Reading, MA, 1967.

[8] F. Della Croce, M. Ghirardi, and R. Tadei. An improved branch-and-bound algorithm for the two machine total completion time flow shop problem. *European Journal of Operational Research*, 139(2):293 – 301, 2002.

[9] F. Della Croce, V. Narayan, and R. Tadei. The two-machine total completion time flow shop problem. *European Journal of Operational Research*, 90(2):227 – 237, 1996.

[10] Boris Detienne, Stéphane Dauzère-Pérès, and Claude Yugma. An exact approach for scheduling jobs with regular step cost functions on a single machine. *Computers & Operations Research*, 39(5):1033–1043, 2012.

[11] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[12] Anis Gharbi, Talel Ladhari, Mohamed Kais Msakni, and Mehdi Serairi. The two-machine flowshop scheduling problem with sequence-independent setup times: New lower bounding strategies. *European Journal of Operational Research*, 231(1):69–78, November 2013.

[13] Mohamed Haouari and Mohamed Kharbeche. An assignment-based lower bound for a class of two-machine flow shop problems. *Computers and Operations Research*, 40(7):1693 – 1699, 2013.

[14] Han Hoogeveen and Tsuyoshi Kawaguchi. Minimizing total completion time in a two-machine flowshop: Analysis of special cases. *Mathematics of Operations Research*, 24(4):887=910, 1999.

[15] Han Hoogeveen, Linda van Norden, and Steef van de Velde. Lower bounds for minimizing total completion time in a two-machine flow shop. *Journal of Scheduling*, 9(6):559–568, 2006.

[16] J.A. Hoogeveen and S.L. van de Velde. Stronger lagrangian bounds by use of slack variables: Applications to machine scheduling problems. *Mathematical Programming*, 70(1-3):173–190, 1995.

[17] Toshihide Ibaraki and Yuichi Nakamura. A dynamic programming method for single machine scheduling. *European Journal of Operational Research*, 76(1):72–82, 1994.

[18] Edward Ignall and Linus Schrage. Application of the branch and bound technique to some flow-shop scheduling problems. *Operations Research*, 13(3):400–412, 1965.

[19] Gio K. Kao, Edward C. Sewell, and Sheldon H. Jacobson. A branch, bound, and remember algorithm for the $1|r_i|\sum T_i$ scheduling problem. *Journal of Scheduling*, 12(2):163–175, August 2008.

[20] Laurent Peridy, Eric Pinson, and David Rivreau. Using short-term memory to minimize the weighted number of late jobs on a single machine. *European Journal of Operational Research*, 148(0):591–603, 2003.

[21] Hanif D. Sherali and Churlzu Lim. Enhancing Lagrangian Dual Optimization for Linear Programs by Obviating Nondifferentiability. *INFORMS J. on Computing*, 19(1):3–13, January 2007.

[22] Hanif D. Sherali and Osman Ulular. A primal-dual conjugate subgradient algorithm for specially structured linear and convex programming problems. *Applied Mathematics and Optimization*, 20(1):193–221, July 1989.

[23] Shunji Tanaka. Extension of the dynasearch to the two-machine permutation flowshop scheduling problem (in Japanese). *Transactions of the Institute of Systems, Control and Information Engineers*, 24(2):23–30, 2011.

[24] Shunji Tanaka, Shuji Fujikuma, and Mituhiko Araki. An exact algorithm for single-machine scheduling without machine idle time. *Journal of Scheduling*, 12(6):575–593, 2009.

[25] V. T'kindt, F. Della Croce, and C. Esswein. Revisiting Branch and Bound Search Strategies for Machine Scheduling Problems. *Journal of Scheduling*, 7(6):429–440, November 2004.

[26] Steef van de Velde. Minimizing the sum of the job completion times in the two-machine flow shop by Lagrangian relaxation. *Annals of Operations Research*, pages 257–268, 1990.