A Novel Integer Linear Programming Formulation for Job-Shop Scheduling Problems

SUBMISSION DATE / POSTED DATE

01-03-2021 / 03-03-2021

CITATION

Liu, Anbang; Luh, Peter; Yan, Bing; Bragin, Mikhail (2021): A Novel Integer Linear Programming Formulation for Job-Shop Scheduling Problems. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.14135075.v1

DOI

10.36227/techrxiv.14135075.v1

A Novel Integer Linear Programming Formulation for Job-Shop Scheduling Problems

Anbang Liu, Peter B. Luh, *Life Fellow*, Bing Yan, *Member*, *IEEE*, Mikhail A. Bragin, *Member*, *IEEE*

***Abstract* – Job-shop scheduling is an important but difficult problem arising in low-volume high-variety manufacturing. It is usually solved at the beginning of each shift with strict computational time requirements. For fast resolution of the problem, a promising direction is to formulate it in an Integer Linear Programming (ILP) form so as to take advantages of widely available ILP methods such as Branch-and-Cut (B&C). Nevertheless, computational requirements on ILP methods for existing ILP formulations are high. In this paper, a novel ILP formulation is presented. In the formulation, a set of binary indicator variables indicating whether an operation begins at a time slot on a machine group or not is selected as decision variables, and all constraints are innovatively formulated based on this set of variables. For fast resolution of large problems, our recent decomposition-and-coordination method "Surrogate Absolute-Value Lagrangian Relaxation" (SAVLR) is enhanced by using a 3-segment piecewise linear penalty function, which more accurately approximates a quadratic penalty function as compared to an absolute-value function. Testing results demonstrate that our new formulation drastically reduces the computational requirements of B&C as compared to our previous formulation. For large problems where B&C has difficulties, near-optimal solutions are efficiently obtained by using the enhanced SAVLR under the new formulation.**

***Index terms*–Manufacturing, job-shop scheduling, integer linear programming, decomposition and coordination**

I. Introduction

Job shops are manufacturing systems designed for low-volume/high-variety production [1]. In a job shop, machines are grouped based on their functionalities, and each group has limited capacities. A part may need to go through a sequence of operations, each can be processed by one or a few machine groups. A schedule is usually generated with strict computational time requirements, e.g., 10 to 20 minutes, at the beginning of a shift. The scheduling problem is subject to four types of constraints: part-to-machine assignment constraints, processing time requirements, operation precedence constraints, and machine capacity constraints. To have on-time delivery – the ultimate goal of job-shop scheduling, the objective function should be due date related, e.g., weighted tardiness penalties. The problem is difficult because of its combinatorial nature.

For fast resolution of job-shop scheduling problems, a promising direction is to formulate them in an Integer Linear Programming (ILP) form so as to take advantages of widely available ILP methods such as Branch-and-Cut (B&C) [2]–[4]. Nevertheless, the efficiency of ILP methods is significantly affected by problem formulations, and when a formulation contains large numbers of decision variables and constraints, the methods may experience difficulties [5]. Developing good formulations is thus of critical importance. This, however, is difficult in view of the many types of complicated constraints within job-shop scheduling problems.

As will be reviewed in Section II, an ILP formulation was presented in [6] based on our classic formulation [7], with recent extensions also reported in [8] and [9]. In [6], operation beginning times were selected as integer decision variables. Processing time requirements and operation precedence constraints were easily formulated based on them. To consider machine capacity constraints, an additional set of binary indicator variables was created to indicate the status of operations: if an operation is active on a machine group at a time slot (i.e., being processed), then the corresponding indicator variable equals one; and zero otherwise. These indicator variables depend on operation beginning times, and to describe such relationships, a large number of constraints is needed. Consequently, when considering large problems, the computational requirements of ILP methods are high.

To overcome the above-mentioned difficulties, a novel ILP formulation will be developed in Section III. In the formulation, a set of binary indicator variables indicating whether an operation begins at a time slot on a machine group or not is selected as decision variables. If an operation begins at a certain time slot on a machine group, then the corresponding indicator variable equals one; and zero otherwise. Based on these variables, all constraints, including machine capacity constraints, are innovatively formulated without introducing additional decision variables or constraints. The numbers of decision variables and constraints are significantly reduced as compared to those of [6]. Computational requirements of B&C are thus drastically reduced, as will be supported through numerical testing in Section V.

Since our new formulation has low computational requirements, B&C can solve small- or medium-sized problems efficiently. For large problems, B&C may still suffer from poor performance. Our recent decomposition-and-coordination method "Surrogate Absolute-Value Lagrangian Relaxation" (SAVLR) is enhanced in Section IV. SAVLR exploits exponential reduction of complexity upon problem decomposition, and effectively coordinates subproblem solutions with accelerated convergence [10]. In the method, machine capacity constraints, which couple various parts

This work is supported in part by National Innovation Center of High Speed Train R&D project "Modeling and comprehensive intelligent optimization for new high efficiency urban rail transit system" under grant No. CX/KJ-2020-0006.

Anbang Liu is with the Center for Intelligent and Networked System (CFINS), Department of Automation, Tsinghua University, Beijing 10084, China (e-mail: liuab19@mails.tsinghua.edu.cn).

Peter B. Luh and Mikhail A. Bragin are with the Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT 06269-4157, USA (e-mail: peter.luh@uconn.edu and mikhail.bragin@uconn.edu.).

Bing Yan is with the Department of Electrical and Microelectronic Engineering at Rochester Institute of Technology, Rochester, NY 14623, USA (e-mail: bxyeee@rit.edu).

assigned to a machine group, are "relaxed." Instead of using powerful quadratic penalty functions as in Augmented Lagrangian Relaxation [11], violations of these constraints are penalized by using an absolute-value function. This absolute-value function is piece-wise linear with two segments, and is exactly linearized by introducing additional decision variables and constraints so as to use ILP solvers. The absolute-value function, however, is a poor approximation to a quadratic function, and the quadratic growth of penalties cannot be well captured. When the level of constraint violation is large, the absolute-value penalty function may not impose a sufficiently large penalty, especially at the early stage of optimization. To overcome this difficulty, the absolute-value penalty function is enhanced by using a 3-segment piecewise linear function, where the middle segment is a constant. This function better captures the quadratic growth of penalties as compared to the absolute-value function. In addition, it can be exactly linearized, and the numbers of additional decision variables and constraints are the same as those required by the absolute-value function. Extension to multiple-segment piecewise linear penalty functions is possible.

To demonstrate the performance of our new formulation and the enhanced SAVLR (or SAVLRE for simplicity), three examples are presented in Section V. The first and the second are for small- and medium-sized instances, respectively, and are solved by using B&C. The results demonstrate that our new formulation drastically reduces the computational requirements of B&C as compared with the original formulation of [6], and optimal solutions are efficiently obtained. The third example with four large instances is solved by using SAVLRE. Testing results demonstrate that the convergence of multipliers is improved by using the new 3-segment penalty function as compared to using the absolute-value penalty function, while computational requirements of each iteration are similar. Near-optimal solutions are therefore obtained in a computationally efficient manner.

II. Literature Review

In this section, existing job-shop scheduling formulations are reviewed in subsection A. Solution methodologies are then discussed in subsection B.

A. Problem formulations

With complicated constraints, formulating a job-shop scheduling problem is difficult [1]. Integer Linear Programming (ILP) formulations were developed in [12]–[17]. In these formulations, the objective is usually to minimize the makespan, i.e., the time span from the very beginning to the end when all the parts are processed. The makespan, however, does not capture the on-time delivery performance, the paramount measure for job-shop scheduling. In fact, due dates are not even included in these formulations. Also, operations are assigned to individual machines, and overlapping of two operations on a machine is prohibited. In view that there are many possible sequences among the operations assigned to a machine, the number of such constraints are large. Also, practically, there are multiple machines of the same functionality in a job shop. The consideration of groups of machines with the same functionality is therefore a better modeling approach.

In [7], one of our earlier papers, a separable nonlinear formulation was presented. Operation beginning times were selected as integer decision variables, indicating when each operation is to begin on an eligible machine group (as opposed to begin on an eligible machine). The objective is to minimize the total weighted tardiness. Machine capacity constraints limit the number of active operations (i.e., operations that are being processed) on a group. Consider for example a machine group with five machines available at a particular time slot. Then at most five operations can be active at that time slot. These machine capacity constraints that couple parts together and the objective function are part-wise additive. After machine capacity constraints are relaxed, the problem can be decomposed into part-wise subproblems, each with much-reduced complexity.

In our recent work [6], an ILP formulation was developed based on [7]. In the formulation, operation beginning times were selected as integer decision variables. From these variables, processing time requirements and operation precedence constraints were easily formulated. The fundamental difficulty is the modeling of machine capacity constraints. An additional set of binary indicator variables was created to indicate if an operation is active or not on an eligible machine group at each time slot following the idea of [7]. Then the sum of these indicators over all relevant operations for a machine group should be less than or equal to the number of machines in that group available at each time slot. This set of indicator variables is related to operation beginning times. To describe such relationships, a significant number of additional constraints are needed. Therefore, although this formulation is linear, B&C might suffer from poor performance for large problems. Very recently, the formulation in [6] was tightened through a systematic approach by transforming the constraints to directly delineate the problem convex hull in [8], and the computational efficiency of B&C has thus been improved.

B. Solution methodologies

In this subsection, heuristics, branch-and cut, standard Lagrangian Relaxation, Surrogate Lagrangian Relaxation and Surrogate Absolute-Value Lagrangian Relaxation are briefly reviewed to solve job shop scheduling problems.

Heuristics. Meta-heuristics such as Particle swarm [18], [19], Tabu search [20], [21] and evolutionary algorithms [22], [23] are frequently used to solve job-shop scheduling problems. These methods have the advantage of low computational requirements. However, convergence is difficult to guarantee, and the quality of solutions is difficult to quantify.

Branch-and-Cut (B&C). Branch-and-Cut (B&C) has been used to solve ILP job-shop scheduling problems, e.g., [6], [15], [16]. The key idea of B&C is to find the convex hull of the problem through adding "valid cuts." If the convex hull is found, then the optimal solution can be obtained by using Linear Programming (LP) methods. If the method fails to obtain the convex hull, or valid cuts are difficult to obtain or are ineffective, then time-consuming Branch-and-Bound is used. Since finding convex hulls itself is NP-hard, B&C may suffer from difficulties for large job-shop scheduling problems.

Standard Lagrangian Relaxation. Another approach is Lagrangian Relaxation (LR) when applying to separable formulations such as that of [7]. In the method, after relaxing

coupling machine capacity constraints, the "relaxed problem" is decomposed into smaller subproblems, one for each part. Since subproblems have much-reduced sizes, their complexity is drastically reduced as compared to that of the original problem. Subproblem solutions are then coordinated through iterative updating of multipliers based on subgradients. However, the stepsizes used to update multipliers require the knowledge of the optimal dual value, which is not available. Although adaptive estimations of the optimal dual value are used in practice, these estimations are typically ineffective and may result in an excessive number of iterations. Moreover, the computational requirements of traditional LR are high because all subproblems need to be solved in order to update multipliers, and multipliers may suffer from the zigzagging difficulties in view of the characteristics of dual functions for ILP problems. Therefore, the performance of traditional LR is poor.

Surrogate Lagrangian Relaxation and Surrogate Absolute-Value Lagrangian Relaxation. By exploiting a contraction mapping concept, our recent "Surrogate Lagrangian Relaxation" (SLR) ([11]) was developed without requiring the knowledge of the optimal dual value to calculate stepsizes. Moreover, multipliers are updated without requiring all subproblems to be solved, thereby reducing the heavy computational requirements and the multiplier zigzagging issues. Thus, most of the major difficulties of traditional LR have been overcome. Convergence is further improved in the "Surrogate Absolute-Value Lagrangian Relaxation" (SAVLR) method in [10], where absolute-value penalty functions are used to penalize the levels of machine capacity violations in [6]. This penalty function can be exactly linearized, and subproblems are solved by using B&C.

III. A Novel Integer Linear Formulation

In this section, a novel ILP formulation for job-shop scheduling problems based on that of [6] is presented.

Consider a job shop with M machine groups. The capacity of machine group m at time slot t is denoted as $M_{m,t}$ for $m \in [1, 2, .., M]$ and $t \in [1, 2, ..., T]$, where T is the total number of time slots and is assumed to be long enough to process all the parts. There are I parts, each with an arrival time a_i and a due date d_i. Part $i \in [1, 2, ..., I]$ needs to go through a sequence of J_i operations, and the j^{th} operation of part i is denoted as (i, j) – a part-operation pair. Let the set of all part-operation pairs be denoted as S. An operation can be processed by one of the eligible machine groups $U_{i,j}$, and the processing time of operation (i, j) on machine group m is denoted as $p_{i,j,m}$, which may be machine group dependent. It is assumed that processing cannot be interrupted, i.e., non-preemptive. In the following, decision variables are first introduced. Then four types of constraints, including part-to-machine assignment constraints, processing time requirements, operation precedence constraints, and machine capacity constraints, are formulated, followed by the objective function.

a) Decision variables

As reviewed in subsection II.A, operation beginning times are integer decision variables in the formulations of [6], [8], [9]. The fundamental difficulty of these formulations lies in the modeling of machine capacity constraints, where an additional set of binary indicator variables was created to indicate whether an operation is active on a machine group at each time slot or not, resulting in large numbers of additional variables and constraints. To overcome this difficulty, a set of binary indicator variables indicating whether an operation begins at a time slot on a machine group or not is selected as decision variables. If operation (i, j) is to begin on machine group m at time slot t, then $b_{i,j,m,t}$ equals one; and it equals zero otherwise. With this innovative selection of decision variables, part-to-machine assignment constraints, i.e., each operation must be assigned to a unique machine group and to begin at a unique time slot, can be easily formulated. Processing time requirements and operation precedence constraints can also be delineated following those in [6], [8], [9]. More importantly, machine capacity constraints can be effectively modeled without introducing additional decision variables or constraints. In the following, these constraints are introduced.

b) Part-to-machine assignment constraints

Since each operation must be assigned to a unique machine group and to begin at a unique time slot, part-to-machine assignment constraints are modeled as follows:

$$\sum_{\forall m \in U_{i,j}} \sum_{t=l_{i,j}}^{u_{i,j}} b_{i,j,m,t} = 1, \forall (i,j) \in S. \tag{1}$$

In the above, the range $[l_{i,j}, u_{i,j}]$ contains the set of eligible operation beginning times for operation (i, j). Specifically, an operation cannot begin too early until there is enough time to complete all the proceeding operations, and it cannot begin too late so that there is not enough time to complete it and its subsequent operations. The method to calculate $[l_{i,j}, u_{i,j}]$ will be described later in (9) and (10).

c) Processing time requirements

From the operation beginning indicator variables, integer operation beginning times are obtained as

$$b_{i,j} = \sum_{\forall m \in U_{i,j}} \sum_{t=l_{i,j}}^{u_{i,j}} t \cdot b_{i,j,m,t}, \tag{2}$$

where the integer variable $b_{i,j}$ is the beginning time of operation (i, j). Since processing is non-preemptive, the completion time equals its beginning time plus the required processing time following equation (2) in [6], i.e.,

$$c_{i,j} = \sum_{\forall m \in U_{i,j}} \sum_{t=l_{i,j}}^{u_{i,j}} (t + p_{i,j,m}) b_{i,j,m,t} - 1, \tag{3}$$

where $c_{i,j}$ is the completion time of operation (i, j). Please note that $\{b_{i,j}\}$ and $\{c_{i,j}\}$ are introduced here for easy understanding and presentation. They are not decision variables and are not optimized in the solution process to be introduced in Section IV.

d) Operation precedence constraints

For each part, its operations need to be processed in a given sequence. Without loss of generality, operations for a part are numbered according to their precedence, and operation $(i, j+1)$ cannot begin until operation (i, j) is completed, i.e.,

$$b_{i,j+1} > c_{i,j}, \forall (i,j) \in \{(i,j) \mid (i,j) \in S, (i,j+1) \in S\}. \tag{4}$$

Equation (4) can be re-written without $\{b_{i,j}\}$ or $\{c_{i,j}\}$ as:

$$\sum_{\forall m \in U_{i,j}} \sum_{t=l_{i,j}}^{u_{i,j}} t \cdot b_{i,j+1,m,t} \geq \sum_{\forall m \in U_{i,j}} \sum_{t=l_{i,j}}^{u_{i,j}} (t + p_{i,j,m}) b_{i,j,m,t}, \tag{5}$$
$$\forall (i,j) \in \{(i,j) \mid (i,j) \in S, (i,j+1) \in S\}.$$

e) Machine capacity constraints

To formulate machine capacity constraints, it is noticed that if operation (*i, j*) is active on machine group *m* at time slot *t*, then its beginning time must be within the interval [*t*-$p_{i,j,m}$+1, *t*]. Therefore, the status (active or not) of operation (*i, j*) on machine group *m* at time *t*, represented by $\delta_{i,j,m,t}$, can be obtained by summing up the operation beginning indicator variables over [*t*-$p_{i,j,m}$+1, *t*]:

$$\delta_{i,j,m,t} = \sum_{\forall k \in [t-p_{i,j,m}+1,t] \cap [l_{i,j},u_{i,j}]} b_{i,j,m,k} \ . \tag{6}$$

Again, variables $\{\delta_{i,j,m,t}\}$ are introduced for easy understanding and presentation. They are not decision variables and are not optimized in the solution process.

Based on (6), the number of active operations on machine group *m* at time slot *t* is obtained by summing up the statuses of all relevant operations that can be processed by the machine group. The machine capacity constraints can therefore be formulated as:

$$\sum_{(i,j)\in O_m} \sum_{\forall k \in [t-p_{i,j,m}+1,t] \cap [l_{i,j},u_{i,j}]} b_{i,j,m,k} \le M_{m,t}, \forall t, \forall m, \tag{7}$$

where O_m denotes the set of operations that can be processed by machine group *m*.

f) The Objective function

Following [6], [7], the objective function to be minimized is the total weighted tardiness. The tardiness of part *i* is the number of time slots being late, i.e., the number of time slots that the completion time of the last operation of part *i* exceeds the due date d_i. It is thus described by $\max(c_{i,J_i} - d_i, 0)$, where (i, J_i) is the last operation of part *i*. The total weighted tardiness is then formulated by summing up the weighted tardiness over all parts:

$$f(c) \equiv \sum_i w_i \cdot \max(c_{i,J_i} - d_i, 0), \tag{8}$$

where w_i is the weight or the importance of part *i*. Equation (8) can be easily re-written without $\{b_{i,j}\}$ or $\{c_{i,j}\}$.

g) Ranges of operation beginning times

To reduce the decision space, possible beginning time slots of operation (i,j), i.e., $[l_{i,j}, u_{i,j}]$, need to be delineated. The values $l_{i,j}$ and $u_{i,j}$ are derived at the data-preprocessing stage as follows. The earliest beginning time slot of the first operation of part *i* is the part arrival time, i.e., $l_{i,1} = a_i$. A subsequent operation cannot begin until there is enough time to complete all the proceeding operations, i.e.,

$$l_{i,j} = a_i + \sum_{k=1}^{j-1} \min_{m \in U_{i,j}} (p_{i,k,m}), \forall j \ge 2. \tag{9}$$

There is a minimization in (9) since processing times may depend on the selections of machine groups, which cannot be predetermined. The smallest processing times are used to give the maximal flexibility in selecting the beginning time slot. Similarly, an operation cannot begin too late so that there is not enough time to complete it and its subsequent operations. We therefore have:

$$u_{i,j} = T - \sum_{k=j}^{J_i} \min_{m \in U_{i,j}} (p_{i,k,m}), \forall j. \tag{10}$$

Here, the smallest processing times are also used.

Based on the above, the job-shop scheduling problem can thus be described as:

$$\min_b \left\{ \sum_i w_i \cdot \max\Big(\sum_{\forall m \in U_{i,J_i}} \sum_{t=l_{i,J_i}}^{u_{i,J_i}} (t + p_{i,J_i,m}) b_{i,J_i,m,t} - 1 - d_i, 0 \Big) \right\}, \tag{11}$$

s.t.(1), (5), (7).

The above objective function with the nonlinear maximization operation can be linearized by introducing additional non-negative integer decision variables $\{z_i\}$ following page 150 of [24] as:

$$\min_{b,z} \left\{ \sum_i w_i \cdot z_i \right\}, \tag{12}$$

subject to constraints (1), (5), (7) and additional inequalities

$$\sum_{\forall m \in U_{i,J_i}} \sum_{t=l_{i,J_i}}^{u_{i,J_i}} (t + p_{i,J_i,m}) b_{i,J_i,m,t} - 1 - d_i \le z_i, \forall i. \tag{13}$$

In view that part-to-machine assignment constraints (1) and operation precedence constrains (5) are only associated with individual parts, and machine capacity constraints (7) that couple operations together and the objective function (8) are part-wise additive, the above formulation is separable.

IV. Solution Methodology

As will be demonstrated in Section V, B&C can solve small- or medium-sized problems based on the new formulation. For large problems, B&C may still suffer from difficulties because of the combinatorial nature of the problem. For fast resolution of such problems, Surrogate Absolute-Value Lagrangian Relaxation (SAVLR) [10] is enhanced in this section.

In SAVLR, the problem is decomposed into subproblems by relaxing the coupling machine capacity constraints (7); and subproblem solutions are coordinated by iteratively updating multipliers. After the machine capacity constraints (7) are relaxed by using Lagrangian multipliers $\{\lambda_{t,m}\}$, the relaxed problem at iteration *k* is formed as:

$$\min_{z,b,s} \left\{ \begin{array}{l} \sum_i w_i \cdot z_i + \sum_t \sum_m \lambda_{t,m}^k \left(g_{t,m}(b) + s_{t,m} \right) + \\ c^k \sum_t \sum_m \left| g_{t,m}(b) + s_{t,m} \right| \end{array} \right\}, \tag{14}$$

s.t. (1), (5), (13),

where

$$g_{t,m}(b) \equiv \sum_{(i,j)\in O_m} \sum_{\forall k \in [t-p_{i,j,m}+1,t] \cap [l_{i,j},u_{i,j}]} b_{i,j,m,k} - M_{m,t}, \tag{15}$$

and $\{s_{t,m}\}$ are non-negative slack variables introduced to convert inequality constraints (7) to equality constraints. In (14), the absolute-value penalty function with the positive penalty coefficient *c* is used instead of a quadratic penalty function as in the Augmented Lagrangian Relaxation method ([11]) to facilitate the use of B&C. It is piecewise linear with two segments, and can be exactly linearized by introducing additional decision variables and constraints. However, since the absolute-value function is a poor approximation to a quadratic function, the quadratic growth characteristics of quadratic penalty functions cannot be well captured. When multipliers are far away from their optimal values and the levels

of constraint violations are large, especially at the early stage of optimization, the absolute-value penalty function might not be able to impose a sufficiently large penalty.

To overcome the above-mentioned difficulty, our idea is to use a convex piecewise linear function with three segments to approximate the quadratic function. In Figure 1, a quadratic function is depicted in blue, and the absolute-value function is depicted in black. It can be seen that the absolute-value function is not a good approximation to the quadratic function. To improve approximation accuracy, take the following convex piecewise linear function with three segments as an example:

$$p(x) \equiv \max(0, 4x-3, -4x-3), \tag{16}$$

which is depicted in red in the figure. As can be seen, with three segments, the approximation accuracy is improved as compared to that of the absolute-value function, in particular for the interval [-4, 4]. For problem instances considered in Section V, the capacity violation of a machine group at a time slot is generally less than 4 according to our testing. Therefore, the function (16) provides a good approximation to the quadratic function, and imposes sufficiently large penalties when the level of violation is 2 or above as compared to those imposed by the absolute-value function.

A convex piecewise linear function such as (16) can be exactly linearized through introducing additional decision variables and constraints following the standard way discussed on page 150 of [24]. To linearize such a function, one additional decision variable is required, and the number of additional constraints equals the number of non-zero segments. For our 3-segment function (16), the number of non-zero segments is two. Therefore, the numbers of additional decision variables and constraints required is equal to those required by the absolute-value function. For other types of problems, the vertices, slopes and the number of segments of the function can be adjusted.

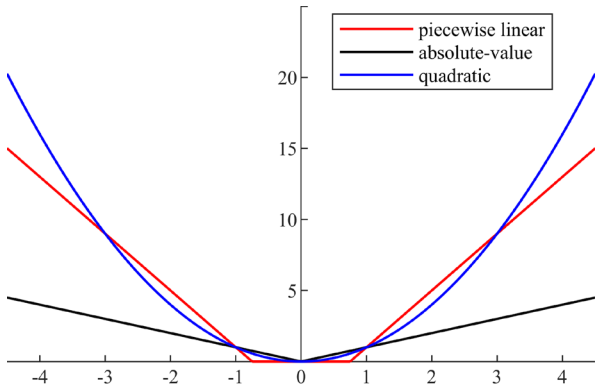

Fig. 1 Illustration of the 3-segment piece-wise linear function, the absolute-value function, and the quadratic function

With the new penalty function (16) replacing the absolute-value penalty function, the solution process follows exactly that of [10]. The relaxed problem becomes:

$$\min_{z,b,s} \left\{ \begin{array}{l} \sum_i w_i \cdot z_i + \sum_t \sum_m \lambda_{t,m}^k \left(g_{t,m}(b) + s_{t,m} \right) + \\ c^k \sum_t \sum_m \max \left(\begin{array}{l} 0, 4\left(g_{t,m}(b) + s_{t,m} \right) - 3, \\ -4\left(g_{t,m}(b) + s_{t,m} \right) - 3 \end{array} \right) \end{array} \right\}, \tag{17}$$

$$s.t. \quad (1), (5), (13).$$

After introducing additional decision variables and constraints to linearize the penalty function, the relaxed problem (17) can be exactly linearized as

$$\begin{aligned} &\min_{z,b,s,r} \left\{ \sum_i w_i \cdot z_i + \sum_t \sum_m \lambda_{t,m}^k \left(g_{t,m}(b) + s_{t,m} \right) + c^k \sum_t \sum_m r_{t,m} \right\}, \\ &s.t. \quad (1), (5), (13), \\ &4\left(g_{t,m}(b) + s_{t,m} \right) - 3 \le r_{t,m}, \forall t, m, \\ &-4\left(g_{t,m}(b) + s_{t,m} \right) - 3 \le r_{t,m}, \forall t, m, \end{aligned} \tag{18}$$

where $\{r_{t,m}\}$ is the additional set of non-negative decision variables introduced by the linearization process. A subproblem for part i can then be formed from (18) by optimizing with respect to decision variables associated with that part while fixing decision variables associated with other parts at their previously obtained values $\{\tilde{b}_{i,j,m,t}\}$. Specifically, it is:

$$\begin{aligned} &\min_{z,b,s,r} \left\{ \begin{array}{l} w_i \cdot z_i + c^k \sum_t \sum_m r_{t,m} + \\ \sum_t \sum_m \lambda_{t,m}^k \left(\sum_{j:(i,j)\in O_m} \sum_{\forall k \in [t - p_{i,j,m}+1, t] \cap [l_{i,j}, u_{i,j}]} b_{i,j,m,k} + s_{t,m} \right) \end{array} \right\}, \\ &s.t. \quad (1), (5), (13), \\ &4\left(g_{t,m}^i(b) + s_{t,m} \right) - 3 \le r_{t,m}, \forall t, m, \\ &-4\left(g_{t,m}^i(b) + s_{t,m} \right) - 3 \le r_{t,m}, \forall t, m, \end{aligned} \tag{19}$$

with

$$\begin{aligned} g_{t,m}^i(b) \equiv &\sum_{j:(i,j)\in O_m} \sum_{\forall k \in [t - p_{i,j,m}+1, t] \cap [l_{i,j}, u_{i,j}]} b_{i,j,m,k} + \\ &\sum_{(i',j)\in O_m : i' \ne i} \sum_{\forall k \in [t - p_{i',j,m}+1, t] \cap [l_{i',j}, u_{i',j}]} \tilde{b}_{i',j,m,k} - M_{m,t}. \end{aligned} \tag{20}$$

Since subproblem (19) has a much reduced dimension, solving it is much easier than solving the original problem.

After solving a subproblem consisting one or a few parts at iteration k by using B&C, multipliers are updated based on surrogate subgradients to coordinate subproblem solutions as:

$$\lambda_{t,m}^{k+1} = \lambda_{t,m}^k + s^k \left(g_{t,m}(b) + s_{t,m} \right), \forall t, m. \tag{21}$$

To guarantee the convergence of multipliers, the stepsize s^k in (21) is updated following equations (18) and (19) in [10]. When the penalty coefficient c^k is too large, the surrogate optimality condition (equation (14) in [10]) may not be satisfied, and solutions may get trapped at a local minimum. In this case, penalty coefficient c^k is decreased following (21) in [10]. Finally, when the stepsize s^k reduces below a certain threshold or when the CPU time reaches a pre-specified limit, the iterative multiplier updating process stops.

Since machine capacity constraints (7) are relaxed, subproblem solutions, when put together, generally do not satisfy (7). Subproblem solutions are thus "repaired" by using heuristics, e.g., the one embedded within solver CPLEX. The idea is to optimize the decision variables associated with violated machine capacity constraints while fixing the remaining decision variables by using B&C. Specifically, if the

capacity of machine group m_1 is violated at time slot t_1, then the parts assigned to m_1 to begin within the range

$$\psi = [t_1 - \beta_1, t_1 + \beta_2], \tag{22}$$

where β_1 and β_2 are small integer values, are selected as decision variables to satisfy (7). The remaining variables are fixed at their latest available values. Subproblem solutions are also repaired when the norm squared of constraint violations is less than or equal to a threshold γ, i.e.,

$$\sum_{t,m} \left(g_{t,m}(b) + s_{t,m} \right)^2 \le \gamma . \tag{23}$$

This is to obtain multiple feasible solutions so that the one with the minimal cost will be selected as the solution.

To measure the quality of feasible solutions, a lower bound is obtained as follows. Suppose at convergence, the final set of multipliers, the penalty coefficient, and subproblem solutions are obtained. All subproblems are solved again to optimality for the given set of multipliers and the penalty coefficient. If the resulting subproblem solutions are the same as those obtained at convergence, then the corresponding surrogate dual value is a dual value, and provides a lower bound to feasible costs.

V. Numerical Results

The new formulation and solution methodology are implemented by using MATLAB R2018a academic version and IBM CPLEX 12.8.0.0. Three examples are tested on a laptop with the Intel Xeon W-10855M processor with six cores at 4.3-Ghz, 64GB of RAM at 2933-Mhz, and Windows 10. The first example is a small instance considered in [6]–[9], and the second example consists of several medium-sized problems. These two examples are solved by using B&C to demonstrate that our new formulation drastically reduces computational requirements of B&C as compared to the original formulation [6]. The third example consists of several large instances, and is solved by using our enhanced SAVLR (or SAVLRE for short) to demonstrate that near-optimal solutions can be efficiently obtained.

Example 1: A small problem

This example consists of an instance with 127 parts, and is taken from Pratt & Whitney's Development Operation shop solved in [6]–[9] to test problem formulations. The problem is solved by using B&C where optimization stops when the computational time reaches 3,600 seconds or the optimal solution is found. The feasible cost, the MIP gap, and the solving time are presented in Table I. As can be seen, the optimal solution is efficiently obtained after 3.31s for the new formulation. For comparison purposes, the existing formulation [6] is also tested. A cost of 15,117 with a gap of 3.72% is obtained after 3,600s. The new formulation is thus much more efficient for B&C to solve than the existing one.

Table Ⅰ Comparison of formulations: small size

New formulation (B&C)			Formulation [6] (B&C)		
Cost[1]	GAP[2]	Solving time (s)	Cost[1]	GAP[2]	Solving time (s)
14,872	0	3.31	15,117	3.72%	3600

1: Total weighted tardiness

2: MIP gap reported by the CPLEX solver

As reviewed in Section II, the formulation of [6] has been improved in [8] by using a systematic formulation tightening approach. As reported in Table V of [8], after tightening, a feasible cost with 0.01% MIP gap was obtained by using B&C after 14.7s. The results thus fall within the range obtained by using our new formulation, and demonstrates the effectiveness of the formulation tightening approach. Tightening of our new formulation will be a future topic for investigation.

Example 2: Medium-sized problems

This example consists of three instances of 200, 250 and 300 parts each. There are 20 machine groups, each with three to four machines. Each part requires one to seven operations, and each operation can be processed by one to three machine groups. The arrival time of each part is generated by using a uniform distribution $U[1, 300]$, and the due date of each part equals its possible earliest completion time (i.e., its arrival time plus its smallest total processing time). As for tardiness weights, 30% of parts have a weight of 1, 65% of parts have a weight of 10, and 5% of parts have a weight of 100. The total number of time slots is 500, which is large enough to process all the parts. The stopping criteria are the same as in Example 1.

The first instance is with 200 parts. By using B&C, the feasible cost, the MIP gap, and the solving time are presented in Table Ⅱ. As can be seen, the optimal solution is efficiently obtained after 12.77s. For comparison purposes, the existing formulation is also tested. No feasible solution can be obtained after 3,600s. The new formulation is thus much more efficient for B&C to solve than the existing one.

The second instance is with 250 parts. As shown in Table II, the optimal solution is efficiently obtained after 33.91s for our new formulation. For the third instance with 300 parts, the optimal solution is obtained after 179.27s. With the existing formulation, no feasible solution can be obtained after 3,600s for both instances.

Table II Comparison of formulations: medium size

Instance	New formulation (B&C)			Formulation [6] (B&C)		
	Cost[1]	GAP	Solving time (s)	Cost	GAP	Solving time (s)
200*20[2]	377	0	12.77	no feasible solution is found after 3600 s		
250*20	593	0	33.91	no feasible solution is found after 3600 s		
300*20	609	0	179.27	no feasible solution is found after 3600 s		

1: Total weighted tardiness

2: 200*20 means the instance with 200 parts and 20 machine groups

To test the robustness of our new formulation with respect to non-constant capacities of machine groups, each machine is assumed to have a probability of 0.3% to break down at each time slot, and repairing takes four consecutive time slots. Based on this, ten breakdown scenarios are randomly generated, and other data are the same as those for the 300-part instance. The optimal solutions of all ten scenarios are efficiently obtained by using B&C. The average solving time is 319s, the minimal solving time is 166s, the maximal solving time is 890s, and the standard deviation is 214s. The results demonstrate the robustness of our new formulation with respect to non-constant machine capacities.

Example 3: Large problems

This example consists of two instances of 500 and 600 parts with 10 machine groups, and two instances of 400 and 500 parts with 20 machine groups. The parts are grouped to form 10 subproblems. Other data are generated following the method of Example 2 without machine breakdowns. The problems are solved by using our enhanced SAVLR (SAVLRE) with the new formulation. When the norm squared of machine capacity violations is less than or equal to 40, i.e., γ in (23) is 40, heuristics are used to find feasible solutions to the original problem, and the parameters β_1 and β_2 in (22) are equal to 5 and 1, respectively. The algorithm stops when the CPU time reaches 1,800s or the stepsize reduces below 0.05.

The first instance is with 500 parts and 10 machine groups. By using the SAVLRE, a solution with a cost of 4,210 is obtained after 813s, and another with cost of 3,942 after 1100s. The lower bound obtained is 3,659. The duality gap is thus 13.1% and 7.1%, respectively. The results demonstrate that high-quality solutions can be efficiently obtained by using SAVLRE. For comparison purposes, B&C is also tested. Since the problem instance is large, no feasible solution is obtained after 3,600s. The feasible cost, the gaps, and the CPU times of both SAVLRE and B&C presented above are summarized in the first row of Table III.

To examine the effects of the new penalty function, the norm squared of machine capacity violations at each minor iteration (i.e., after solving one subproblem) is depicted in red in Figure 2. For comparison purposes, those of SAVLR are depicted in blue. As can be seen, the new method enjoys a faster reduction of the machine capacity violations than SAVLR. This is because our new 3-segments penalty function imposes larger penalties at the early stage of optimization.

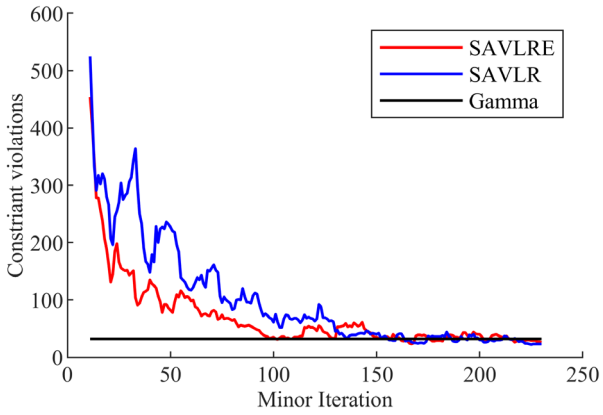

Fig. 2 the norm squared of constraint violations in each minor iteration

According to our testing, at almost all iterations, the capacity violation of each machine group at each time slot is less than 4 for both SAVLR and SAVLRE. Therefore, our 3-segmnent function (16) provides a good approximation to the quadratic function and sufficiently large penalties. Moreover, since the numbers of additional decision variables and constraints required by linearization are the same with those required by the absolute-value function, the computational requirements are similar with those of the absolute-value penalty function. CPU time per minor iteration is 3.91s on average for SAVLRE, and 3.52s for SAVLR. Testing of 5- and 7-segment piecewise linear penalty functions has also been conducted, and the results are not satisfactory. This is because linearizing these functions requires too many additional constraints, resulting in significantly increases of subproblem solving times.

To demonstrate the scalability of SAVLRE, three more instances are tested, and the results are also shown in Table III. As can be seen, near-optimal solutions are effectively obtained by using SAVLRE, but not by using B&C.

Table III Comparison of methodologies: large problems

Instance	New formulation (SAVLRE+B&C)				New formulation (B&C)		
	Cost[1]	Lower Bound	GAP	CPU time (s)	Cost	GAP	CPU time (s)
500*10[2]	3942	3659	7.1%	1100	no feasible solution is found after 3600 s		
600*10	7448	7008	5.9%	1261	no feasible solution is found after 3600 s		
400*20	2019	1907	5.5%	404	2773	30.91%	3600
500*20	3967	3554	10.4 %	872	no feasible solution is found after 3600 s		

1: Total weighted tardiness

2: 500*10 means the instance with 500 parts and 10 machine groups

VI. Conclusion

In this paper, a novel ILP formulation is developed for job-shop scheduling, resulting in a drastic reduction of computational requirements for B&C as compared to the formulation of [6]. SAVLR is also enhanced so that near-optimal solutions can be efficiently obtained for large problems. These advancements will have major implications on formulating and resolution of other manufacturing scheduling problems and beyond. To further improve solution quality and computation efficiency, the future work will include the tightening of the new formulation.

References

[1] M. Pinedo, *Scheduling*, vol. 29. Springer, 2012.

[2] M. Padberg and G. Rinaldi, "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems," *SIAM review*, vol. 33, no. 1, pp. 60–100, 1991.

[3] R. Gomory, "An algorithm for the mixed integer problem," RAND CORP SANTA MONICA CA, 1960.

[4] R. E. Gomory, "Solving linear programming problems in integers," *Combinatorial Analysis*, vol. 10, pp. 211–215, 1960.

[5] L. A. Wolsey and G. L. Nemhauser, *Integer and combinatorial optimization*, vol. 55. John Wiley & Sons, 1999.

[6] B. Yan, M. A. Bragin, and P. B. Luh, "Novel formulation and resolution of job-shop scheduling problems," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3387–3393, 2018.

[7] D. J. Hoitomt, P. B. Luh, and K. R. Pattipati, "A practical approach to job-shop scheduling problems," *IEEE transactions on Robotics and Automation*, vol. 9, no. 1, pp. 1–13, 1993.

[8] B. Yan, M. Bragin, and P. Luh, "An Innovative Formulation Tightening Approach for Job-Shop Scheduling," 2021, TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.12783893.v1

[9] B. Yan, M. A. Bragin, and P. B. Luh, "Tightened Formulation and Resolution of Energy-Efficient Job-Shop Scheduling," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, 2020, pp. 741–746.

[10] M. A. Bragin, P. B. Luh, B. Yan, and X. Sun, "A scalable solution methodology for mixed-integer linear programming problems arising in automation," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 2, pp. 531–541, 2018.

[11] D. P. Bertsekas, *Nonlinear programming*, Third Edit. Athena Scientific, Belmont, MA, 2016.

[12] J. C.-H. Pan and J.-S. Chen, "Mixed binary integer programming formulations for the reentrant job shop scheduling problem," *Computers & Operations Research*, vol. 32, no. 5, pp. 1197–1212, 2005.

[13] C. Özgüven, Y. Yavuz, and L. Özbakır, "Mixed integer goal programming models for the flexible job-shop scheduling problems with separable and non-separable sequence dependent setup times," *Applied Mathematical Modelling*, vol. 36, no. 2, pp. 846–858, 2012.

[14] V. Roshanaei, A. Azab, and H. ElMaraghy, "Mathematical modelling and a meta-heuristic for flexible job shop scheduling," *International Journal of Production Research*, vol. 51, no. 20, pp. 6247–6274, 2013.

[15] M. Karimi-Nasab and M. Modarres, "Lot sizing and job shop scheduling with compressible process times: A cut and branch approach," *Computers & Industrial Engineering*, vol. 85, pp. 196–205, 2015.

[16] S. Chansombat, P. Pongcharoen, and C. Hicks, "A mixed-integer linear programming model for integrated production and preventive maintenance scheduling in the capital goods industry," *International Journal of Production Research*, vol. 57, no. 1, pp. 61–82, 2019.

[17] L. Meng, C. Zhang, Y. Ren, B. Zhang, and C. Lv, "Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem," *Computers & Industrial Engineering*, vol. 142, p. 106347, 2020.

[18] G. Zhang, X. Shao, P. Li, and L. Gao, "An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem," *Computers & Industrial Engineering*, vol. 56, no. 4, pp. 1309–1318, 2009.

[19] M. Nouiri, A. Bekrar, A. Jemai, S. Niar, and A. C. Ammari, "An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem," *Journal of Intelligent Manufacturing*, vol. 29, no. 3, pp. 603–615, 2018.

[20] S. Dauzère-Pérès and J. Paulli, "An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search," *Annals of Operations Research*, vol. 70, pp. 281–306, 1997.

[21] M. Saidi-Mehrabad and P. Fattahi, "Flexible job shop scheduling with tabu search algorithms," *The international journal of Advanced Manufacturing technology*, vol. 32, no. 5–6, pp. 563–570, 2007.

[22] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 193–208, 2013.

[23] X.-N. Shen and X. Yao, "Mathematical modeling and multi-objective evolutionary algorithms applied to dynamic flexible job shop scheduling problems," *Information Sciences*, vol. 298, pp. 198–224, 2015.

[24] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.