LINEAR PROGRAMMING-BASED ALGORITHMS FOR THE MINIMUM MAKESPAN HIGH MULTIPLICITY JOBSHOP PROBLEM

MICHAEL MASIN
IBM RESEARCH – HAIFA,
MOUNT CARMEL, HAIFA 31905, ISRAEL

AND

TAL RAVIV
DEPARTMENT OF INDUSTRIAL ENGINEERING, TEL AVIV UNIVERSITY,
TEL AVIV 69978, ISRAEL

Abstract. We study a generalized version of the minimum makespan jobshop problem in which multiple instances of each job are to be processed. The system starts with specified inventory levels in all buffers and finishes with some desired inventory levels of the buffers at the end of the planning horizon. A schedule that minimizes the completion time of all the operations is sought.

We develop a polynomial time asymptotic approximation procedure for the problem. That is, the ratio between the value of the delivered solution and the optimal one converge into one, as the multiplicity of the problem increases. Our algorithm uses the solution of the linear relaxation of a time indexed Mixed Integer formulation of the problem. In addition, a heuristic method inspired by this approximation algorithm is presented and is numerically shown to out-perform known methods for a large set of standard test problems of moderate job multiplicity.

Keyword: Optimization, Jobshop, Approximations, Heuristics

1. Introduction

1.1. **Background.** The minimum makespan jobshop problem is a notoriously hard combinatorial optimization problem. It is known to be NP-Hard already for two machines instances (See, Lenstra and Rinnooy-Kan (1979)) and for instances with three machines and three jobs (Sotskov and Shakhlevich (1995)). Williamson et al. (1997) showed that the problem is not approximable within 5/4. The problem is approximable within $O(\log^2 m)$ when m is the number of machines, if each job should be processed on each of the machines at most once (Shmoys et al. (1994)). Goldberg et al. (1997) present a polynomial time approximation algorithm for the general problem within

$$O\left(\lceil \min(mO_{max}, P_{max})/\log\log(mO_{max})\rceil \log(mO_{max})/\log\log(mO_{max})\right)$$

when O_{max} denotes the maximal number of operations for a single job and P_{max} denotes the maximal processing time of an operation. Sevastjanov was the first to introduce a geometric approximation algorithm that guaranty a solution value within a constant of the optimum, independent of the number of jobs. This result was first reported in the western literature in Sevast'janov (1994).

Several specialized and generic meta-heuristics methods were applied to the minimum makespan jobshop problem. For a survey on application of genetic Algorithms see Blackstone et al. (1982). An

Date: September 2013.

application of the Tabu search meta-heuristics is reported in Nowicki and Smutnicki (1996). Adams et al. (1988) introduced a successful specialized heuristic, called the *shifting bottleneck*.

The fluid-based approximation paradigm was specifically designed for large instances of complex scheduling problems with high multiplicity, see Dai and Weiss (1996), Bertsimas and Gamarnik (1999), Dai and Weiss (2002), Boudoukh et al. (2001), Bertsimas and Sethuraman (2002) and Bertsimas et al. (2003). Recently, a new paradigm of approximation algorithms, referred as linear programming (LP) based algorithm was introduced and applied for simpler scheduling problems. For literature on LP-based approximation scheduling see, e.g., Goemans et al. (2002), Chekuri and Khanna (2004), Savelsbergh et al. (1998), Skutella (2006), Correa and Wagner (2005) and references therein. Both approaches derive dispatching rules from the solutions of continuous relaxations of scheduling problems. The algorithm presented in this paper builds on these two paradigms.

The high multiplicity jobshop is a variant of the jobshop model that allows multiple instances of each job class. An asymptotically optimal algorithm, in this context, is one that provides an approximation ratio that approaches one as the multiplicity of the jobs approaches infinity. Fluid-based heuristics are based on the important observation that the scheduling of instances with a large number of items that belong to few classes can be approximated by the optimal solution of a similar, but much simpler, fluid control problem. Several asymptotically optimal algorithms based on this idea for the minimum makespan high multiplicity jobshop problem have been developed. Bertsimas and Gamarnik (1999) presented an approximation algorithm with a sub-linear error term. Bertsimas and Sethuraman (2002) and Boudoukh et al. (2001) devised fluid-based heuristics with an additive error term. In practical terms, multiplicity of several hundred is enough for obtaining near-optimal schedules using the above mentioned algorithms.

Brauner et al. (2005) pointed out that the analysis of the complexity of algorithms for high multiplicity scheduling problems requires special care because their output may be exponential in size.They proposed a complexity framework for this class.

Closely related literature deals with creating cyclic schedules for a set of jobs (sometimes also referred to as *periodic* or *high-multiplicity scheduling*) assuming the number of part sets to schedule is infinite. The objective in this line of literature is typically to minimize cycle time and/or the average flow time. In this setting the transient behavior of the system is unimportant and the solution can be described in terms of the schedule of a single cycle. For similar earlier studies, see Lee and Posner (1997), Hall et al. (2002), and Leung et al. (2004). Kimbrel and Sviridenko (2008) shows that the problem is APX-hard even for instances with a single class of jobs with unit time operations and provides an approximation algorithm with approximation ratio that depends on the problem parameters for this problem. Amin et al. (2011) and Kechadi et al. (2013) present heuristic methods based on simulated annealing neural network, respectively, to solve the cyclic jobshop problem.

1.2. **Problem definition and main results.** In this paper we study a more general high multiplicity jobshop model, in which in addition to multiplicity of jobs, the system may start and end with some prespecified initial and final inventory levels. Consequently, the multiplicity of operations may vary across operations of the same job. We denote this model by $J/multi, s_{r,o}, e_{r,o}/C_{max}$, where $s_{r,o}$ represent the fact that intermediate inventory levels at various production steps may be initially found in the system; $e_{r,o}$ represent the intermediate inventory that should be built by the end of the planning horizon. The notation of $s_{r,o}$ and $e_{r,o}$ is rigorously defined below. We note that this is a reentrant settings in which jobs may visit the same machine several times. The objective is minimum makespan, i.e., to minimize the completion time of the last operation. The main contributions of this study are:

1. Development of an efficient asymptotic approximation LP-based algorithm for the high multiplicity jobshop problem with an additive error of $O_{max}U_{max}$ where O_{max} denotes the number of operations in the longest processing route and U_{max} denotes the maximal total processing time of a single instance of all operations on a given machine. This improves the additive error of $O_{max}(U_{max}+2P_{max})$[1] achieved by the Fluid Synchronization Algorithm (FSA) of Bertsimas and Sethuraman (2002), in which P_{max} denotes the processing time of the longest operation. However, it should be noted that the FSA procedure runs in linear time with respect to the multiplicity of the jobs, while our procedure is much more demanding and its running time is bounded by a polynomial in the total length of the operations. A significant and novel component of our algorithm is a careful mathematical program formulation that enables us to borrow some ideas from the analysis of Bertsimas and Sethuraman (2002).
2. Presentation and analysis of an α-point family of LP-based algorithms and a heuristic method based on these algorithms. This heuristic is numerically shown to improve significantly over existing algorithms for jobshop problems with moderate multiplicity. In our benchmark instances, more than 80% of the optimality gap is closed on average with a minimal improvement of closing 50% of the gap. In addition, while the absolute gap of the existing algorithms is unaffected by the multiplicity, our heuristic significantly reduces the absolute gap as the multiplicity grows. At multiplicity 10 the absolute gap of the proposed algorithm is, on average, six times smaller than the absolute gap of the existing algorithm.
3. Introduction of the first asymptotic approximation algorithm for the high multiplicity jobshop problem with initial and final inventory of work in process. This feature makes our algorithm useful in practical scenarios in which the manufacturing system operates continuously and decisions are made in a rolling horizon manner. Consider, for example, a jobshop that accept orders for new jobs every day and needs to schedule these jobs in addition to previous jobs that are already completed some of their operations. Allowing planning for the final inventory (of semi-finished jobs) may be useful in a hybrid make-to-stock setting when one does not wish to complete the production until more information about the properties (such as color) of the desired final products is known. The analysis of this extension preformed in a straight forward manner in our LP-approximation setting but may be more complex for previous methods such as the FSA.

The rest of the paper is organized as follow. In Section, 2, we provide a Mixed Integer Linear Programming (MILP) formulation for the minimum makespan high multiplicity jobshop problem and study its LP relaxation. The LP-based Synchronization Algorithm (LPSA) is introduced and analyzed in Section 3. The dispatching rule derived in this section is based on the start time of each operation in the LP-relaxed problem. In Section 4, LPSA is extended into a family of algorithms that is parameterized by an arbitrary α-point. In Section 5, we propose a scheduling heuristic based on LPSA. This heuristic is numerically tested and compared with competing methods and with lower bounds in Section 6. Finally, we discuss our results and sketch some ideas for further research in Section 7.

[1]We present the results of Bertsimas and Sethuraman (2002) using our notation here for the sake of comparison. In their notation, U_{max} is bounded from above by the number of job classes (I) times the maximal processing time P_{max} and $J_{max} \equiv O_{max}$.

2. MILP Formulation and Partial Relaxation

In this section we present a time-indexed mixed-integer linear programming (MILP) formulation of the $J/multi, e_{r,o}, s_{r,o}/C_{max}$ problem. We then define a *partial* LP relaxation of it, which relaxes most but not all of its integrality constraints. The partially relaxed model is shown to be solvable in polynomial time. We use the following notations in the mathematical models below:

Indices

- r Product classes (routes)
- o Operations
  The pair (r, o) denotes the o^{th} operation of the r^{th} product class. The buffer (r, o) refers to the buffer that holds items that are waiting for operation (r, o).
- i Machine
- t Time periods

Constants

- T The number of (discrete) time periods that constitute the planning horizon. T is assumed to be sufficiently large for the completion of all the operations in an optimal solution of the minimal makespan problem
- I Set of machines
- R Set of product classes
- O_r Set of operations of product class r
- $p_{r,o}$ The time (integer number of periods) required to carry out a single instance of operation (r, o)
- σ_i The set of operations carried out on machine i
- $\sigma(r, o)$ The machine that processes operation type (r, o)
- $s_{r,o}$ Initial (start) buffer level of operation (r, o).
- $e_{r,o}$ Desired (end) buffer level of operation (r, o).
- U_i Sum of the processing time of all the operation types on machine i, $U_i = \sum_{(r,o)\in\sigma_i} p_{r,o}$.

The elements of each set S are named $1, ..., |S|$. To simplify the notation, assume that any reference to an element outside the valid range is omitted from the actual program and replaced with zero.

Decision variables

- $x_{r,o,t}$ Binary variable that equals one if an instance of operation (r, o) is started on its machine at time period t.
- $y_{r,o,t}$ Number of jobs in buffer (r, o) at period t (after jobs that are starting to be processed at this period were taken from the buffer but before jobs that finished their pervious operation at the end of this period were placed into the buffer), it is defined for each period $t = 0, ..., T + 1$ where $y_{r,o,0} = s_{r,o}$ is the initial level and $y_{r,o,T+1} = e_{r,o}$ is the final level of the buffer of operation (r, o).
- z_t Binary variable that denote whether the system is active at period t.

We are now ready to present a MILP formulation of the high multiplicity jobshop problem. A relaxation of this formulation will serve as the basis to our LP-based Synchronization Algorithm (LPSA).

MILP1

$$\min \sum_{t=1}^{T} z_t \tag{1}$$

$$z_t \geq z_{t+1} \quad \forall t = 1, ..., T \tag{2}$$

$$\sum_{(r,o)\in\sigma_i} \sum_{\tau=t-p_{r,o}+1}^{t} x_{r,o,\tau} \leq z_t \quad \forall t = 1, ..., T,\ i \in I \tag{3}$$

$$y_{r,o,t-1} + x_{r,o-1,t-p_{r,o-1}} - x_{r,o,t} = y_{r,o,t} \quad \forall r \in R, o \in O_r, t = 1, ..., T+1 \tag{4}$$

$$y_{r,o,0} = s_{r,o} \quad \forall r \in R, o \in O_r \tag{5}$$

$$y_{r,o,T+1} = e_{r,o} \quad \forall r \in R, o \in O_r \tag{6}$$

$$x_{r,o,t} \in \{0,1\} \quad \forall r \in R, o \in O_r, t = 1, ..., T \tag{7}$$

$$z_t \in \{0,1\} \quad \forall t = 1, ..., T \tag{8}$$

$$y_{r,o,t} \geq 0 \quad \forall r \in R, o \in O_r,\ t = 1, ..., T \tag{9}$$

Our objective (1) is to minimize the total number of active periods. Together with constraint (2) this is equivalent to minimizing makespan. Constraint (3) limits the processing capacity of each machine and relates z_t to the actual processing start times represented by the x's. Constraint (4) is an inventory balance constraint that updates the buffer level, y. Constraints (5) and (6) define boundary conditions for initial and final inventory levels of the buffers. The remaining constraints define the domains of x, z and y. Note that the integrality of y is implied. The classical minimum makespan jobshop problem is obtained as a special case of this model when $e_{r,o} = 0$ for all the operations (r, o), $s_{r,o} = 0$ for all operations with $o > 2$, and $s_{r,1} = 1$ for all product classes. We note that to solve the above MILP (and its relaxation below) an upper bound on the required number of periods T is needed - we will address this issue in the proof of Proposition 2.2 below.

Next we consider a partial LP relaxation of this MILP, obtained by replacing constraint (7) with

$$x_{r,o,t} \geq 0 \quad \forall r \in R, o \in O_r,\ t = 1, ...T, \tag{10}$$

but keeping (8), the binarity constraint for z_t. We call this new program **MILP2**. Then we define a new constraint $p'_{r,o} \geq p_{r,o}$. We will assume $p'_{r,o} = U_{\sigma(r,o)}$ until the end of Section 3. Additionally, we define

$$\delta_{r,o} \equiv \sum_{q=1}^{o-1} p'_{r,q}. \tag{11}$$

The number of instances of operations (r, o) to be carried out throughout the planning horizon is

$$n_{r,o} = \sum_{q=1}^{o} (s_{r,q} - e_{r,q}). \tag{12}$$

Recall that the initial inventory level at the first buffer of each route ($s_{r,1}$) represents the amount of raw material available for the system. A negative value of $n_{r,o}$ represents an infeasible instance of the problem. The total working time to process all operations of type (r, o) is $n_{r,o}p_{r,o}$. The congestion of the system is defined as

$$\hat{C} = \max_i \sum_{(r,o)\in\sigma_i} n_{r,o}p_{r,o}. \tag{13}$$

This is an obvious lower bound on the minimum makespan problem defined by MILP1. The next inequality provides an upper bound on the minimum makespan.

Next we tighten MILP2 by adding the constraint

$$x_{r,o,t} = 0 \qquad \forall t > \hat{C} + \delta_{r,o} \tag{14}$$

and replacing the inventory balance constraint (4) with the following constraint, which we call the *coordination constraint*,

$$y_{r,o,t-1} + x_{r,o-1,t-p'_{r,o-1}} - x_{r,o,t} = y_{r,o,t} \quad \forall r \in R, o \in O_r, t = 1, ..., T. \tag{15}$$

This creates a new program called **MILP3**. Recall that according to our convention, the term $x_{r,o-1,t-p'_{r,o-1}}$ for $t \leq p_{r,o-1}$ and for $o = 1$ are omitted, since they refer to negative index values. We argue that MILP3 is indeed more constrained than MILP2 in the sense defined by the following proposition.

Proposition 2.1. *The optimal solution of MILP2 is not greater than the optimal solution of MILP3.*

Proof. We show that for any feasible solution of MILP3, and in particular for the optimal one, constructing a solution with the same objective function value for MILP2 is possible. Recall that the objective functions of these two programs are identical and consist of the sums of the z variables. Next, we rewrite constraints (4) and (5) in MILP2 as

$$y_{r,o,t} = s_{r,o} + \sum_{\tau=1}^{t-p_{r,o-1}} x_{r,o-1,\tau} - \sum_{\tau=1}^{t} x_{r,o,\tau} \qquad \forall r \in R, o \in O_r, t = 0, ..., T. \tag{16}$$

This equations are obtained by summation of all the corresponding constraints (4) and (5) up to each period t and thus equivalent. Similarly, constraints (5) and (15) of MILP3 can be rewritten as

$$y_{r,o,t} = s_{r,o} + \sum_{\tau=1}^{t-p'_{r,o-1}} x_{r,o-1,\tau} - \sum_{\tau=1}^{t} x_{r,o,\tau} \qquad \forall r \in R, o \in O_r, t = 0, ..., T. \tag{17}$$

Now consider a feasible solution $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ of MILP3 and calculate new values for $\mathbf{y}$ using constraint (16) based on the obtained values of $\mathbf{x}$. We denote these values by $\tilde{\mathbf{y}}$. Now we complete the proof by showing that $(\mathbf{x}, \tilde{\mathbf{y}}, \mathbf{z})$ is feasible for MILP2. Note that constraints (2), (3), (7), and (8) are satisfied because they are identical in MILP2 and MILP3 and involve only the $\mathbf{x}$ and $\mathbf{z}$ variables. Now constraints (4) and (5) are satisfied by $(\mathbf{x}, \tilde{\mathbf{y}})$ since $\tilde{\mathbf{y}}$ in calculate by equation (16) that equivalent to these constraints. Our only remaining task is showing that the desired end-level constraints (6) and the non-negativity constraints of the y variables (9) are satisfied.

To prove the nonnegativity of $\tilde{y}$, note that the difference between equations (16) that determine the value of the $\tilde{y}$ variables and equations (17) that define the y variables is that the right hand side of the former includes several additional elements of the $x_{r,o,t}$ variables. Specifically, this difference is

$$\sum_{\tau=t-p'_{r,o-1}}^{t-p_{r,o-1}} x_{r,o-1,\tau}.$$

This difference must be non-negative because $x_{r,o,t} \geq 0$. Therefore $\tilde{y}_{r,o,t} \geq y_{r,o,t} \geq 0$ for all (r, o, t).

Finally, to show that constraint (6) is satisfied, we have to show that $(\mathbf{x}, \tilde{\mathbf{y}}, \mathbf{z})$ the inventory end levels obtained by equation (16) at period $T + 1$ equals $e_{r,o}$. To this end, we show that for the same value of $x_{r,o,t}$, the right hand side of equations (16) and (17) are equal. Indeed, the difference between the two equations is

$$\sum_{\tau=T-p'_{r,o-1}}^{T-p_{r,o-1}} x_{r,o-1,\tau}.$$

However, by equation (14) of MILP3, the value of these terms must be zero. ■

Proposition 2.2.

$$OPT(MILP3) \leq \hat{C} + \max_r \left\{\delta_{r,|O_r|} + p_{r,|O_r|}\right\}.$$

Proof. We construct a feasible solution for MILP3, with $T = \hat{C} + \max_r \left\{\delta_{r,|O_r|} + p_{r,|O_r|}\right\}$ as follows

$$x_{r,o,t} = \begin{cases} \frac{n_{r,o}}{\hat{C}}, & 1 + \delta_{r,o} \leq t \leq \hat{C} + \delta_{r,o} \\ 0, & \text{otherwise.} \end{cases} \tag{18}$$

$z_t = 1$ for all $t = 1....,T$. Once the x's are known, the values of the y's are uniquely determined by equations (5) and (15) so it is left to show that constraints (3), (6) and (9) are satisfied.

We first note that for all operations (r, o) and time periods t,

$$\sum_{\tau=t-p_{r,o}+1}^{t} x_{r,o,\tau} \leq p_{r,o}\frac{n_{r,o}}{\hat{C}} \leq \frac{p_{r,o}n_{r,o}}{\sum_{(r',o')\in\sigma_{(r,o)}} n_{r',o'}p_{r',o'}}.$$

The last inequality follows from the definition of $\hat{C}$ in (13). Next, by summing the above inequality over all operations carried out by each machine we see that the capacity constraint (3) is satisfied.

$$\sum_{(r,o)\in\sigma_i} \sum_{\tau=t-p_{r,o}+1}^{t} x_{r,o,\tau} \leq 1 \qquad \forall i \in I, t = 1, ..., T.$$

To prove the feasibility of our solution with respect to the non-negativity constraint (9) we rewrite equality (15) as follows

$$y_{r,o,t} = s_{r,o} + \sum_{\tau=1}^{t-p'_{r,o-1}} x_{r,o-1,\tau} - \sum_{\tau=1}^{t} x_{r,o,\tau}. \tag{19}$$

By (18) we have

$$\sum_{\tau=1}^{t-p'_{r,o-1}} x_{r,o-1,\tau} = \sum_{\tau=1+\delta_{r,o}-p'_{r,o-1}}^{t-p'_{r,o-1}} \frac{n_{r,o-1}}{\hat{C}} = (t - \delta_{r,o})^+\frac{n_{r,o-1}}{\hat{C}}$$

and

$$\sum_{\tau=1}^{t} x_{r,o,\tau} = \sum_{\tau=1+\delta_{r,o}}^{t} \frac{n_{r,o}}{\hat{C}} = (t - \delta_{r,o})^+\frac{n_{r,o}}{\hat{C}}$$

in which $x^+ \equiv \max(0, x)$. Next, we can rewrite (19) as

$$y_{r,o,t} = s_{r,o} + (t - \delta_{r,o})^+\frac{n_{r,o-1} - n_{r,o}}{\hat{C}} = s_{r,o} + (t - \delta_{r,o})^+\frac{e_{r,o} - s_{r,o}}{\hat{C}} \tag{20}$$

The last equality follows directly from the definition of $n_{r,o}$ as in (12). Now, to prove that $y_{r,o,t} \geq 0$ it is sufficient to show that

$$s_{r,o} \geq (t - \delta_{r,o})\frac{s_{r,o}}{\hat{C}}$$

which is true because $t - \delta_{r,o} \leq \hat{C}$ that is indeed the case. To understand this, recall that we defined the length of the planning horizon to be $T = \hat{C} + \max_r \left(p_{r,|O_r|} + \delta_{r,|O_r|}\right)$ and by definition $\delta_{r,o} \leq \delta_{r,|O_r|}$.

Finally, equation (6) holds since, from (20), $y_{r,o,t} = e_{r,o}$ for $t = \hat{C} + \delta_{r,o}$ and in the constructed solution $x_{r,o-1,t-p'_{r,o}} = x_{r,o,t} = 0$ for $t > \hat{C} + \delta_{r,o}$.

We established the upper bound by presenting a feasible solution for MILP3 with $T = \hat{C} + \max_r \left\{\delta_{r,|O_r|} + p_{r,|O_r|}\right\}$.
■

Define $U_{\max} \equiv \max_{i \in I} U_i$, $P_{\max} \equiv \max_{(r,o)} p_{r,o}$ and $O_{\max} \equiv \max_{r \in R} \{|O_r|\}$. Then the bound obtained by Proposition 2.2 can be simplified as follows:

$$OPT(MILP3) \leq \hat{C} + (O_{\max} - 1)U_{\max} + P_{\max}.$$

While the feasible solution constructed in the proof of Proposition 2.2 is far from being optimal in many typical instances of the problem, we point out that its gap from the lower bound $\hat{C}$ is independent of the multiplicity of the jobs. That is the constructed solution is an asymptotically optimal solution of MILP3, meaning that its relative gap from the optimal solution approaches one, as the number of jobs in each class increases. In addition, note that this is an asymptotically optimal solution of MILP2, since $\hat{C}$ is a lower bound of this program as well.

3. The LP Synchronization Algorithm

In this section, we define a dispatching rule for a jobshop model based on the optimal fractional solution of MILP3. We show that the makespan of the resulted schedule is within a constant from the optimal one. This constant is independent of the multiplicity of the instance, and therefore our dispatching rule is asymptotically optimal.

In our high multiplicity setting the term operation refers to a class of tasks that are carried out at the same position in the sequence on the same job type. Such tasks are repeated many times and each such repetition is referred to as an *operation instance*. The n^{th} instance of operation (r, o) is indexed by the tuple (r, o, n)

Given a feasible solution of MILP3 we define the following constant **LP Starting Time** for each operation instance (r, o, n):

$$LPS(r, o, n) = \max\left(t \in \{0, ..., T\} : \sum_{\tau=1}^{t} x_{r,o,\tau} \leq n - 1\right)$$

For $n \leq 0$ we set $LPS(r, o, n) = 0$ by convention. In a solution of the discrete system, the (r, o, n) operation instance is said to be *available* at time t, if the number of instances that completed operation $(r, o-1)$ plus $s_{r,o}$ is at least n, but the number of operations (r, o) that started by this time is less than n. We define the LP synchronization algorithm (LPSA) as follows:

The LP Synchronization Algorithm (LPSA): At any time t, when a machine i is ready and some operation instances are available for it, start to process an instance that minimizes $LPS(r, o, n)$ over the set of available instances.

Note that LPSA can be applied to any feasible solution of MILP3, not only to necessarily its optimal solution. Indeed, our polynomial time approximation procedure is based on a feasible solution of this MILP constructed in Proposition 2.2 that can be obtained in negligible time. Both the FSA, introduced by Bertsimas and Sethuraman (2002) and LPSA are based on relaxations of the minimum makespan high multiplicity jobshop problem. However our LP formulation is tighter than the fluid one. We formalize this statement by the following proposition:

Proposition 3.1. *Any feasible solution of MILP3 is also a feasible solution of the fluid model presented in equations (4)-(8) of Bertsimas and Sethuraman (2002).*

Proof. We present the fluid model using our notation, in which $y_{r,o}(t)$ denotes fluid level in the buffer of operation (r,o) at time t and $x_{r,o}(t)$ denotes the fraction of the capacity allocated by machine $\sigma_{r,o}$ to operation (r,o) at time t.

$$\min \int_0^\infty \mathbb{I}\{\exists (r,o) : y_{r,o}(t) \neq e_{r,o}\}\, dt \tag{21}$$

$$y_{r,1}(t) = s_{r,1} - \int_0^t \frac{x_{r,1}(\tau)}{p_{r,1}} d\tau \qquad \forall r \in R, t \tag{22}$$

$$y_{r,o}(t) = s_{r,o} - \int_0^t \frac{x_{r,o}(\tau)}{p_{r,o}} d\tau + \int_0^t \frac{x_{r,o-1}(\tau)}{p_{r,o-1}} d\tau \qquad \forall r \in R, o = 2, ..., |O_r|, t \tag{23}$$

$$\sum_{(r,o)\in\sigma_i} x_{r,o}(t) \leq 1 \qquad \forall i \in I, t \tag{24}$$

$$x_{r,o}(t), y_{r,o}(t) \geq 0 \qquad \forall (r,o), t \tag{25}$$

Note that this model is more general than the one presented in Bertsimas and Sethuraman (2002), since it allows for the specification of initial and final levels of work in process in the system's buffers. The original model is obtained as a special case when $s_{r,o} = 0$ for all the operations (r,o) with $o > 1$ and $e_{r,o} = 0$ for all the operations (r,o). In the general case, finding the optimal may be computationally hard and requires a specialized algorithm, such as the one presented in Weiss (2008).

It is easy to check that the above constraints are satisfied and that the value of the objective function (21) is equal to that of MILP3 with a solution determined by the same $x_{r,o,\tau}$ by selecting $x_{r,o}(t) = \sum_{\tau=\lceil t-p_{r,o}\rceil+1}^{\lceil t\rceil} x_{r,o,\tau}$ and determining $y_{r,o}(t)$ by equations (22) and (23). ■

While both relaxations allow jobs to be divided and workstations to process fractions of different jobs at the same time, the LP relaxation does not allow the same fraction of a job to be processed simultaneously on different machines. In the model of MILP3, each fraction of an operation is delayed for the processing time of the whole operation before it is available for the next one. This property of our LP relaxation provides stronger lower bounds and delivers fractional solutions that are more similar to the discrete solutions and therefore translate more readily into actual schedules. In addition, our LP formulation can be easily extended to accommodate additional constraints and different objective functions, e.g., allowing initial and final work in process.

We refer to the starting time of operation instance (r,o,n) returned by LPSA as *the discrete starting time* of (r,o,n) and we denote it as $DS(r,o,n)$. Similarly, we refer to the completion time of an operation instance (r,o,n) in the schedule obtained by LPSA as the *the discrete completion time* and denote it as $DC(r,o,n)$. That is, $DC(r,o,n) \equiv DS(r,o,n) + p_{r,o}$. For $n \leq 0$ we set $DC(r,o,n) = 0$ by convention. These constants may be used to encode any particular solution. When such a solution is obtained by the LPSA, we have the following proposition:

Proposition 3.2. *A solution obtained by LPSA satisfies*

(A) $LPS(r,o,n) \geq DC(r,o-1,n-s_{r,o})$

(B) $DC(r,o,n) \leq LPS(r,o,n) + U_{\sigma(r,o)}$

for all its operation instances (r,o,n).

Proof. With respect to the given solution, we define the following set of scheduling epochs $0 = t_0 \leq t_1 < t_2 < \cdots$, where each time t_j denotes a time in which one or more operations instances are started or ended. In other words, for each epoch t_j, there are some instances (r, o, n) such that $t_j = DS(r, o, n)$ or $t_j = DC(r, o, n)$.

We prove this proposition by induction on the scheduling epochs of the system. That is, we show that the fact that the proposition holds for all operation instances (r, o, n) such that $LPS(r, o, n) \leq t_{j-1}$ implies it truth for all (r, o, n) instances such that $LPS(r, o, n) \leq t_j$.

For the induction base note that claim (A) holds for the first epoch since at time 0, only operations instances with strictly positive initial inventory $s_{r,o} \geq 1$ can start being processed by MILP3. Claim (B) holds trivially, because no operation is completed by time 0.

To proceed with (A), consider an instance (r, o, n) such that $LPS(r, o, n) \leq t_j$. The claim trivially holds for $n \leq s_{r,o}$. Note that by constraints (5) and (15) of MILP3, $LPS(r, o, n) \geq LPS(r, o-1, n - s_{r,o}) + U_{\sigma(r,o-1)}$ for all $o > 1, n > s_{r,o}$. By (2), $DC(r, o-1, n-s_{r,o}) \leq LPS(r, o-1, n-s_{r,o}) + U_{\sigma(r,o-1)}$, hence we obtain the required inequality $LPS(r, o, n) \geq DC(r, o-1, n-s_{r,o})$.

To prove (B), note that if (A) holds for all (r, o, n), such that $LPS(r, o, n) \leq t_{j-1}$, the claim holds for each machine at any scheduling epoch that occurs in an idle time of the machine, including the one that concludes it and starts the next busy period. Clearly, during an idle time all the instances for which $LPS(r, o, n)$ is due were already started (and completed), otherwise the machine would have been busy at this time in processing them.

We define a *busy period* of a machine in a given schedule as a period that starts and ends at an idle time or at an epoch $\tilde{t}$, such that $\tilde{t} = DS(r, o, n) < LPS(r, o, n)$[2] for some instance (r, o, n). Consider an epoch $t = LPS(r', o', n')$, such that $DS(r', o', n') \geq t$. Time t is either within or at the beginning of a busy period of machine $\sigma(r', o')$. We denote the beginning of this busy period by t'. Let $k_{r,o}$ denote the number of instances of operation (r, o) that started in the continuous system on machine $\sigma(r', o')$ during the interval $[t', t]$. That is, $k_{r,o} = |\{n : t' \leq LPS(r, o, n) \leq t\}|$. Note that for the $k^{th}_{r,o}$ operations to start, at least $\max(k_{r,o} - 1, 0) \equiv (k_{r,o} - 1)^+$ operations of this type must have been started and completed during the interval in the LP solution. Let us denote the set of operations processed by machine $\sigma(r, o)$ by $\Sigma(r, o)$ and let $\Sigma^-(r, o) = \Sigma(r, o) \setminus \{(r, o)\}$. Then,

$$t - t' \geq \sum_{(r,o)\in\Sigma(r',o')} (k_{r,o} - 1)^+ p_{r,o}$$

. This is because in the continuous solution, $p_{r,o}$ units of time are allocated for each instance (r, o, n) that was started and completed during the interval $[t', t]$. Then,

$$\begin{aligned}
DS(r', o', n') &\leq t' + \sum_{(r,o)\in\Sigma^-(r',o')|k_{r,o}\geq 1} k_{r,o}p_{r,o} + \sum_{(r,o)\in\Sigma^-(r',o')|k_{r,o}=0} p_{r,o} \\
&= t' + \sum_{(r,o)\in\Sigma(r',o')} (k_{r,o} - 1)^+ p_{r,o} + \sum_{(r,o)\in\Sigma^-(r',o')} p_{r,o} \\
&\leq t' + t - t' + \sum_{(r,o)\in\Sigma^-(r',o')} p_{r,o} \\
&= t + \sum_{(r,o)\in\Sigma^-(r',o')} p_{r,o} \\
&= LPS(r', o', n') + \sum_{(r,o)\in\Sigma^-(r',o')} p_{r,o}.
\end{aligned}$$

[2]This may happen since, if a machine is ready at time t, LPSA dispatches an available operation instance with the smallest $LPS(r, o, n)$ even if $LPS(r, o, n) > t$

That is, the starting time in an actual discrete system $DS(r', o', n')$ can be delayed by at most $\sum_{(r,o)\in\Sigma^-(r',o')} p_{r,o} = U_{\sigma(r',o')} - p_{r'o'}$ with respect to $LPS(r', o', n')$. Finally, since $DC(r', o', n') = DS(r', o', n') + p_{r',o'}$ claim (B) follows. ■

We denote the makespan obtained from LPSA by $C(LPSA)$. The main result of this section is the following approximation guarantee of LPSA.

Theorem 3.3.

$$C(LPSA) \leq \hat{C} + \max_r \sum_{o\in O_r} U_{\sigma(r,o)} \leq \hat{C} + O_{\max}U_{\max}$$

Proof. By Constraint (14) we conclude that

$$LPS(r, o, n) \leq \hat{C} + \max_r \sum_{o=1}^{|O_r|-1} U_{\sigma(r,o)} \leq \hat{C} + (O_{\max} - 1)U_{\max}$$

for all operations instances (r, o, n) and in particular for ones that determine the makespan of the schedule obtained by LPSA. The claim is then followed by Proposition 3.2 stating that the completion time $DC(r, o, n) \leq LPS(r, o, n) + U_{\sigma(r,o)}$. ■

We point out that this is an *asymptotically optimal* approximation result since the upper bound is not related to the actual number of operations to be performed. Clearly, if the number of operations of each class to be performed by the system is $Kn_{r,o}$, then as $K \to \infty$ the ratio,

$$\frac{C(LPSA)}{\hat{C}} \to 1.$$

The approximation guarantee of Theorem 3.3 dominates the approximation obtained in Bertsimas and Sethuraman (2002) for FSA, which can be stated, using our notation, as

$$C(FSA) \leq \hat{C} + max_r \sum_{o\in O_r} \left(U_{\sigma(r,o)} + 2p_{r,o}\right) \leq \hat{C} + O_{\max}(U_{\max} + 2P_{\max}).$$

The difference between the approximation guarantees of the two methods may be substantial when the number of operation classes carried out on each machine is relatively small or when the processing time of few operation classes in each machine is much greater than the processing times of the rest.

Theorem 3.4. *The asymptotic optimality of the LPSA procedure can be obtained in polynomial time with respect to the size of its output.*

Proof. We note that the approximation guaranty proved by Theorem 3.3 is applicable if LPSA is applied with the feasible solution of MILP3 constructed in the proof of Proposition 2.2. Recall that this solution can be obtained in constant time, independent of the multiplicity and the values of other parameters. Once a feasible solution of MILP3 is known, LPSA determine which operation should be dispatched on machine i, in $O(|\sigma_i|)$ (recall that σ_i is the set of operations on machine i to choose from). ■

4. α-Point Family of LP Synchronization Algorithms

In this section we generalized the LPSA algorithm presented in the previous section into a family of similar *asymptotically optimal* approximation algorithms, in which the dispatching rule is determined by the completion time of some predefined α fraction of the operation instances in the LP relaxation. The proofs in this section are similar to corresponding proofs in previous ones and thus omitted from the text. We show that the best approximation guarantee is obtained for either $\alpha = 0$ or $\alpha = 1$ but it is worth to try using various value of α since the actual best solution can be obtained for any of these values. In Section 5 we will use a similar concept to devise a successful heuristic for the high multiplicity jobshop problem.

Let us define

$$P_i \equiv \max_{(r,o)\in\sigma_i} p_{r,o}$$

and redefine $p'_{r,o}$ as follows:

$$p'_{r,o} \equiv p_{r,o} + (1-\alpha)(U_{\sigma(r,o)} - p_{r,o}) + \alpha P_{\sigma(r,o+1)} = (1-\alpha)U_{\sigma(r,o)} + \alpha(p_{r,o} + P_{\sigma(r,o+1)}), \tag{26}$$

where, $\alpha \in [0,1]$. Note, that when $\alpha = 0$, we have $p'_{r,o} = U_{\sigma(r,o)}$, as in the previous sections.

Now we create a new auxiliary program parameterized by α called $\mathbf{MILP3}_\alpha$ by using $p'_{r,o}$ defined in (26), adding $\alpha(|O_r|-1)$ parts to the initial inventory level of the first buffer for all products r, and redefining Constraint (14). That is, we replace $s_{r,1}$ by $s_{r,1} + \alpha(|O_r|-1)$.

Let us the number of instances of operation (r,o) to be performed in MILP3$_\alpha$ by $n'_{r,o} = n_{r,o} + \alpha(|O_r|-1)$, then the total working time to process them is $n'_{r,o}p_{r,o}$. The congestion of the system in MILP3$_\alpha$ is

$$\hat{C}' = \max_i \sum_{(r,o)\in\sigma_i} n'_{r,o}p_{r,o} = \max_i \sum_{(r,o)\in\sigma_i} \left(n_{r,o} + \alpha(|O_r|-1)\right) p_{r,o} \le \hat{C} + \max_i \sum_{(r,o)\in\sigma_i} \alpha(|O_r|-1)p_{r,o}. \tag{27}$$

Constraint (14) is replaced by Constraint (28) where $\hat{C}$ is replaced by $\hat{C}'$:

$$x_{r,o,t} = 0 \qquad \forall t > \hat{C}' + \delta_{r,o} \tag{28}$$

Clearly, MILP3 is a special case of MILP3$_\alpha$ with $\alpha = 0$. Please recall that $\delta_{r,o}$ defined in Equation (11) will use the new $p'_{r,o}$. Proposition 2.2 is generalized to

Proposition 4.1.

$$OPT(MILP3_\alpha) \le \hat{C} + \max_i \sum_{(r,o)\in\sigma_i} \alpha(|O_r|-1)p_{r,o} + \max_r(\delta_{r,|O_r|} + p_{r,|O_r|}).$$

The bound obtained by Proposition 4.1 can be simplified to the following

$$OPT(MILP3_\alpha) \le \hat{C} + (O_{\max}-1)U_{\max} + P_{\max} + 2\alpha(O_{\max}-1)P_{\max},$$

and in particular, the claim of Proposition 2.2 is obtained as a special case for $\alpha = 0$.

Since in MILP3$_\alpha$ $\alpha(|O_r|-1)$ extra parts of each product r are produced, let us change its solution as follows: for each (r,o) remove first $\alpha(o-1)$ and last $\alpha(|O_r|-o)$ instances, so that exactly $n_{r,o}$ instances of operation (r,o) start. That is,

$$x'_{r,o,t} = 0 \quad \forall r \in R, o \in O_r, t \in \left\{ t : \sum_{\tau=1}^{t} x_{r,o,\tau} \le \alpha(o-1) \text{ or } \sum_{\tau=t}^{T} x_{r,o,\tau} \le \alpha(|O_r|-o) \right\},$$

Now, if in the solution of MILP3$_\alpha$, for some operation (r,o), the first $\alpha(o-1)$ instances are ended in the "middle" of a discretized time unit, that is there exist a pair (r,o) and $t = 1,...,T$ such that

$\sum_{\tau=1}^{t-1} x_{r,o,\tau} < \alpha(o-1)$ but $\sum_{\tau=1}^{t} x_{r,o,\tau} \geq \alpha(o-1)$, we have $x'_{r,o,t} = \sum_{\tau=1}^{t} x_{r,o,\tau} - \alpha(o-1)$. Similarly, if the last $|O_r| - o$ jobs start in a "middle" of a time unit, that is, there exist a pair (r,o) and time t such that $\sum_{\tau=t+1}^{T} x_{r,o,\tau} < \alpha(|O_r|-o)$ but $\sum_{\tau=t}^{T} x_{r,o,\tau} \geq \alpha(|O_r|-o)$, we have $x'_{r,o,t} = \sum_{\tau=t}^{T} x_{r,o,\tau} - \alpha(|O_r|-o)$.

It can be verified that the resulting solution is a feasible solution of MILP3, where the values of the y and z variables are uniquely determined by the values of the x. We refer to this solution as the **adjusted solution** of MILP3$_\alpha$. Next we define the notion of α-points applied to any solution of MILP3$_\alpha$:

The LP α-Point Time: the time when the continuous system completes a fraction α of the n^{th} instance of operation (r,o) $LPQ(r,o,n,0) = LPS(r,o,n) + p_{r,o}$. Formally, $LPQ(r,o,n,0) = LPS(r,o,n) + p_{r,o}$ for $\alpha = 0$ and $LPQ(r,o,n,\alpha) = \min\left(t : \sum_{\tau=1}^{t} x_{r,o,\tau} \geq n - 1 + \alpha\right) + p_{r,o}$ for $\alpha \in (0,1]$.

For $n \leq 0$ we set $LPQ(r,o,n,\alpha) = 0$ by convention. Note that in the original solution of $MILP3_\alpha$, Constraint (15) implies the following constraint on α-points for $n > s_{r,o}$:

$$LPQ(r,o,n,\alpha) - p_{r,o} \geq LPQ(r,o-1,n-s_{r,o},\alpha) - p_{r,o-1} + p'_{r,o-1},$$

whereas in the *adjusted* solution $LPQ(r,o,n,\alpha)$ becomes $LPQ(r,o,n,0)$ or, equivalently,

$$LPS(r,o,n) \geq LPQ(r,o-1,n-s_{r,o},\alpha) + (1-\alpha)(U_{\sigma(r,o-1)} - p_{r,o-1}) + \alpha P_{\sigma(r,o)} \tag{29}$$

Now we are ready to present the LP Synchronization Algorithm parameterized by α (LPSA$_\alpha$).

The LP Synchronization Algorithm (LPSA$_\alpha$): At any time t, when a machine i is ready and some operation instances are available for it, process an available operation instance (r,o,n) that minimizes $LPQ(r,o,n,\alpha)$ of the adjusted solution.

Proposition 4.2. *A solution obtained by $LPSA_\alpha$ satisfies*

(A) $LPS(r,o,n) \geq DC(r,o-1,n-s_{r,o}) + \alpha P_{\sigma(r,o)}$.

(B) $DC(r,o,n) \leq LPQ(r,o,n,\alpha) + (1-\alpha)(U_{\sigma(r,o)} - p_{r,o})$.

for all operation instances (r,o,n).

Let us denote the makespan obtained from $LPSA_\alpha$ by $C(LPSA_\alpha)$. From Constraint (28) and Proposition 4.2 the following approximation guarantee follows directly.

Theorem 4.3.

$$C(LPSA_\alpha) \leq \hat{C} + \max_i \sum_{(r,o)\in\sigma_i} \alpha(|O_r|-1)p_{r,o} + \max_r \left(\sum_{o\in O_r} \left((1-\alpha)U_{\sigma(r,o)} + \alpha p_{r,o}\right) + \alpha \sum_{o=1}^{|O_r|-1} P_{\sigma(r,o+1)} \right)$$
$$\leq \hat{C} + O_{\max}U_{\max} + \alpha\left(2O_{\max}P_{\max} - P_{\max} - U_{\max}\right).$$

The result is a bit surprising—the approximation guarantee is linear in α, i.e., the best approximation guarantee for $LPSA_\alpha$ is obtained for either $\alpha = 0$ or $\alpha = 1$. These two cases may be intuitively explained as follows. If the number of operation classes carried out on the bottleneck machine is relatively small, say less than $(2O_{\max} - 1)$, then we have $\alpha = 0$, otherwise, with many classes on the bottleneck machine and, consequently, $P_{\max}(2O_{\max} - 1) < U_{\max}$, we have $\alpha = 1$.

We point out that the schedule obtained by LPSA$_\alpha$ is also *asymptotically optimal* for every $\alpha \in [0,1]$. Since using α-points can significantly improve LPSA-based heuristics as shown in our numerical experiments below, asymptotic optimality of the whole family is a nice feature to have even though the approximation guaranty for the cases with $0 < \alpha < 1$ is inferior.

5. LPSA Heuristic

In this section we present two heuristic methods that are based on the ideas of the $LPSA_\alpha$ algorithm presented above. In the next section, we show numerically that these methods can deliver a nearly optimal solution for problem instances with moderate multiplicity.

The coordination constraint, (15), seems too restrictive. This is due to the fact that it is typically possible in practice to dispatch each operation instance (r, o, n) long before $(1-\alpha)U_{\sigma(r,o-1)} + \alpha P_{\sigma(r,o)}$ units of time passed since its starting time on the previous machine on its route. An alternative approach, presented in this section, is to solve the partial relaxation MILP2,with the original processing times and number of operations, calculate $LPQ(r, o, n, \alpha)$ using its optimal solution, and apply $LPSA_\alpha$ with various values of α. We refer to this algorithm as LPSA-H1.

In addition, we derived yet another similar heuristic called LPSA-H2, by defining a variant of $LPQ(r, o, n, \alpha)$ as follows: $LPQ'(r, o, n, 0) = LPS(r, o, n)$ and $LPQ'(r, o, n, \alpha) = \min\left(t : \sum_{\tau=1}^{t} x_{r,o,\tau} \geq n - 1 + \alpha\right) + \alpha p_{r,o}$ for $\alpha \in (0, 1]$ based on the same solution of MILP2, i.e., $LPQ'(r, o, n, \alpha) = LPQ(r, o, n, \alpha) - (1 - \alpha)p_{r,o}$. The $LPSA_\alpha$ algorithm can then be applied using various α-point completion times.

We show that MILP2 may be solved by solving a series of (continuous) linear programs. By Constraint (2), the series of binary variables $z_1, ..., z_T$ is non-increasing and thus uniquely defined by the last non-zero element. All the elements up to this one equal one and the rest equal zero. A simple approach to solve MILP2 would be to search for the index of the last non-zero element in $\mathbf{z}$ using a bisection method. For a conjectured value of $T' = \sum_{t=1}^{T} z_t$, we can solve an LP that is identical to MILP2 except that $z_1, ..., z_{T'} = 1$ and $Z_{T'+1}, ..., z_T = 0$. If this LP is feasible, then the minimum makespan is bounded from above by T', and otherwise it is bounded from below by $T' + 1$. For an initial lower bound, we can use $\hat{C}$. An initial upper bound can be obtained by any quick dispatching rule for the high multiplicity jobshop problem. For example, LPSA itself can be used, based on some "smooth" feasible solution that can be obtained in a manner similar to that of the feasible solution of MILP3, which is described in the proof of Proposition 2.2. The value of such a solution is bounded from above by $\hat{C} + \max_r \sum_{o=1}^{|O_r|} p_{r,o} \leq \hat{C} + O_{max}P_{max}$. The total number of calls to the LP by the bisection procedure is $O\left(\log\left(O_{max}P_{max}\right)\right)$. Note that the solution time of each LP is pseudo-polynomial in the size of the input because the number of variables and constraints is exponential in the digit size of the parameters, but polynomial in the values of the parameters. Hence, the overall complexity of this procedure is pseudo-polynomial. We cannot provide an algorithm that dominates the complexity of such a bisection procedure, but we present a heuristic that was shown to be much quicker in our numerical experiments. To this end, we define the following linear program:

LP1

$$v^* = \min \sum_{t=LB+1}^{UB} w_t \cdot z_t \tag{30}$$

$$\sum_{(r,o)\in\sigma_i} \sum_{\tau=t-U_{\sigma(r,o)}+1}^{t} x_{r,o,\tau} \leq 1 \qquad \forall t = 1, ..., LB, i \in I \tag{31}$$

$$\sum_{(r,o)\in\sigma_i} \sum_{\tau=t-U_{\sigma(r,o)}+1}^{t} x_{r,o,\tau} \leq z_t \qquad \forall t = LB+1, ..., UB, i \in I \tag{32}$$

$$0 \leq z_t \leq 1 \qquad \forall t = LB+1, ..., UB \tag{33}$$

and equations $(4) - (10)$

where LB and UB denote currently known lower and an upper bounds.

In addition, we define some series of non-decreasing positive constants $w_{LB+1}, ..., w_{UB}$, e.g., $w_t = t - LB$. Now we can solve MILP2 by solving LP1 iteratively with different values of LB and UB as described below.

Algorithm 1. *Solution Procedure for MILP2*

Initialization: *$LB = \hat{C}$ and UB equals the makespan obtained by some quick heuristic.*

Step 1: *Solve LP1 using the current LB and UB.*

Step 2: *If $v^* = 0$ then*

return current solution, as the optimal solution of MILP2, with $z_t = 1$ for all $t = 1, ..., LB$, $z_t = 0$ for all $t = LB + 1, ..., T$,

exit.

else

$UB = \max\{t : z_t > 0\}$,

$LB = \min\{t : \sum_{\tau=LB+1}^{t} w_\tau \geq v^*\}$,

Goto step1.

We note that the solution of LP1 at Step 1 of Algorithm 1 is amendable to a "warm start", since a dual feasible solution can be constructed based on the optimal solution obtained in the previous iteration. The algorithm 1 terminates in a finite number of iterations, because at each iteration the lower bound is increased by at least one, and the upper bound is not increased. In practice, our tests show that Algorithm 1 terminates in 2-5 iterations when used with weights $1, 2^2, 3^2, ...$

An alternative initial lower bound for the makespan that dominates $\hat{C}$ can be obtain by

$$\hat{C}^+ = \max_{i \in I} \left\{ \sum_{(r,o) \in \sigma_i} n_{r,o} p_{r,o} + \min_{r:(r,o) \in \sigma_i} \sum_{o' < \text{first}(r,i)} p_{r,o'} + \min_{r:(r,o) \in \sigma_i} \sum_{o' > \text{last}(r,i)} p_{r,o'} \right\}$$

where $\text{first}(r,i) := \min\{o : (r,o) \in \sigma_i\}$ and $\text{last}(r,i) := \max\{o : (r,o) \in \sigma_i\}$. The idea behind this lower bound is that the makespan cannot be smaller than the time it takes to process all the operations on a machine plus the time that must pass before this machine may start its first operation and the time that must pass after the last operations on the machine are completed and until other operations of jobs processed by the machine are completed on machines down their routes. Clearly, for problem instances with high multiplicity, $\hat{C}^+$ is relatively close to $\hat{C}$, but it turns out that for instances with moderate multiplicity, $\hat{C}^+$ may be much tighter than the $\hat{C}$, allowing for a reduction in the running time of Algorithm 1.

Note that since MILP2 is an LP relaxation of a valid formulation of the minimum makespan jobshop problem, its optimal solution is a lower bound on the minimum makespan problem, and it is shown to be a strong one.

Our numerical experiments in Section 6 show that the Algorithm 1 is computationally viable for fairly large instances. Larger instances in which MILP2 could not be solved directly using Algorithm 1, can be approximated by dividing the processing times of the operations by some factor, $f > 1$, and by rounding the obtained ratios. We considered two rounding methods, either down or to the nearest integer, with the exception that processing times that are smaller than the factor ar always rounded up to one. The first rounding method yields a valid lower bound for the problem, while the second method sometimes leads to better solutions when used with our heuristics.

The number of non-zeros in LP2 is approximately $T(P+O)$ where T is the number of periods in the planning horizon, $P=\sum_{(r,o)} p_{r,o}$ and $O=\sum_r |O_r|$. Note that if all processing times $p_{r,o}$ are given as integers, $P \geq O$ and typically $P \gg O$. Now by dividing all the processing times by f, the actual makespan is also reduced approximately by a factor of f and hence the problem can be represented by an LP with approximately $\frac{TP}{f^2}+\frac{TO}{f}$ non zeros. That is, the number of non-zeros coefficients is reduced by the order of $O(f^2)$. Assuming that due to processing time and memory limitations we are able to solve a series of linear programs with up to K non zeros in the coefficient matrix, then f should be chosen as the solution of

$$\frac{TP}{f^2}+\frac{TO}{f}=K$$

but not less than one. That is, we use

$$f=\max\left\{1,\frac{TO+\sqrt{T^2O+4KPT}}{2K}\right\}. \tag{34}$$

The solution time of MILP2 is by far the most computationally demanding part of both LPSA-H1 and LPSA-H2. Since both heuristics are based on the same solution of MILP2 applying each one of them for numerous values of α makes sense. In the experiment, describe below, we tried 1001 different quantiles $\alpha=0,0.001,0.002,...,1$.

6. Numerical Experiment

In this section, we present the numerical study we carried out to demonstrate the applicability of our heuristic algorithms. We applied both H1 and H2 methods to a set of 24 OR-Library benchmark jobshop problems, see Beasley (1990). Specifically, we used one 6×6 problem denoted in OR-Lib by 'ft6', thirteen 10×10 instances, denoted by 'ft10', 'abz5', 'abz6', 'orb01'-'orb10', five 20×10 problems, 'swv01'-'swv05', and five 20×15 problems, 'swv06'-'swv10'.

The standard benchmark instances consist of a single job per route. Hence, to create problems with higher multiplicity we duplicated each job several times. We created instances with uniform multiplicity of 1,2,5, and 10. In addition, for the 6×6 and 10×10 instances, we also created instances with variable multiplicity for each job, such that the total number of jobs was 28 and 50, respectively. Overall, we tested 110 different instances. The largest problem instances we solved consisted of hundreds of jobs and thousands of operations to be scheduled. Such problems are much too large for any exact optimization method, or involved search heuristic such as the successful Shifting Bottleneck, introduced by Adams et al. (1988).

We compared the solutions we obtained using the LPSA-H1 and H2 with those obtained using the FSA proposed by Bertsimas and Sethuraman (2002). This algorithm, along with the similar GFA (see Boudoukh et al. (2001)), are, to the best of our knowledge, the only existing algorithms designed specifically for high multiplicity jobshop problems. In addition, we compared our results with schedules obtained by SPT, a simple dispatching rule that may surpass FSA for a few instances of low multiplicity, as shown in Bertsimas and Sethuraman (2002).

LPSA-H1 produced better solutions for 46 out of our 110 test instances. H2 won in 38 cases and in the rest of the instances the two methods tied. It appears that none of the methods dominated the other and we recommend using both of them and selecting the best result. All of the results presented in this section refer to the best results between LPSA-H1 and LPSA-H2.

Our test was conducted on an Intel Core 2 Duo E4500, 2.2GHz with 4GB of RAM. Algorithm 1 was programmed in Ilog-Opl 5.2 and LP1 was coded in this modeling language. The solution of LP1 was

obtained by Ilog-Cplex 10.2 (64bit version). The implementation of Algorithm 1 is "quick and dirty" and in particular we do not use the optimal result of each iteration as an advanced initial basis to warm start the next iteration. Cplex was configured to use its barrier algorithm due to its superior performances for very large linear programs. For the larger instances, the program size had to be reduced by a factor, calculated by (34), as described in the previous section. We used $K = 10^7$ to obtain acceptable solution times of the LP. The LPSA heuristics, as well as FSA and SPT, were programmed using MathWorks Matlab 7.7. The processing times of both FSA and SPT are negligible and our method uses both algorithms to create an initial upper bound for Algorithm 1. Initial lower bounds were obtained by $\hat{C}^+$.

In Tables 1 and 2 we present the values of the solutions obtained by FSA, SPT and the LPSA heuristics along with the obtained lower bounds and some additional information. Table 1 presents the solution values of the instances with fixed multiplicity of $N = 1, 2, 5$, and 10. Consequently, the number of jobs in each route is exactly N. In the first column, we list the names of the instances as they appear in OR-Lib. Exact descriptions of these instances can be found on the OR-Lib web site, see Beasley (1990). The multiplicity is shown in the second column (N), and the table is sorted by it. In the third column, we present $\hat{C}$, the total processing time on the most loaded machine. This simple lower bound is used by Bertsimas and Sethuraman (2002) and Boudoukh et al. (2001) to test their results with FSA and GFA, respectively. The solutions obtained by the FSA and SPT heuristics are presented in the next two columns. In the sixth column, we present the lower bound (LB) obtained by the MILP3 that we solved using algorithm 1. In the seventh and eighth columns we present the solution obtained by our algorithms (LPSA-H1 and LPSA-H2). These are the best out of 1001 values of α used ($\alpha = 0, 0.001, 0.002, ..., 1$). The next columns present information regarding the realization of our algorithm. First, we present the factor that was used to reduce the processing time in the lager instances. A factor of 1 indicates that no factorization was carried out. The next column defines the step when the optimal solution is found by LPSA-H1. In the two rightmost columns, we present the processing times of Step 1 and Step 2, respectively (Step 2 is only relevant for $f > 1$). These times represent almost all the processing times required for solving the problems since the LPSA-H1 and H2 heuristics run, for all possible values of α in negligible time.

In Table 2 we present the same details for problem instances with variable multiplicity of jobs. The multiplicity of "ft06" here is $(8, 4, 3, 1, 3, 9)$ and the multiplicity of the rest of the problems is $(8, 4, 3, 1, 3, 9, 7, 4, 10, 1)$. The structure of this table is identical to that of Table 1 except that the column N, which describe the multiplicity, is omitted.

The performances of our algorithm for the 110 test instances are summarized in Table 3. The first column presents the category of the instances in terms of multiplicity and dimension.The number of instances in each category is shown in parenthesis. Each multiplicity category is divided into subcategories based on the instances dimensions, namely the number of machines and job types. In the second column we present the average improvement in the objective function value of the solutions obtained by our algorithm, as compared to the best between FSA and SPT. That is

$$\text{Solution Improvment} = \frac{\min(FSA, SPT) - \min(H1, H2)}{\min(FSA, SPT)}.$$

The third column presents the average increase of the lower bound that could be obtained using Algorithm 1, as compared to the simple $\hat{C}$ lower bound used by FSA. That is

$$\text{LB Improvment} = \frac{C(LPSA) - \hat{C}}{C(LPSA)}.$$

Name	N	$\hat{C}$ L.B	$\hat{C}^+$ L.B	FSA Solution	SPT Solution	LPSA L.B	$\alpha = 0$ Solution	H1 Solution	H2 Solution	Factor	Opt. Step	Time Step 1	Time Step 2
ft06	1	43	52	65	88	55	70	59	59	1	1	1s	-
abz5	1	868	1000	1467	1352	1135	1363	1281	1281	1	1	3h54	-
abz6	1	688	784	1045	1097	883	1039	980	982	1	1	1h56	-
ft10	1	631	796	1184	1074	859	1090	1044	1081	1	1	1h15	-
orb01	1	643	928	1368	1478	968	1271	1188	1154	1	1	0h50	-
orb02	1	671	733	1007	1175	813	1005	939	937	1	1	1h36	-
orb03	1	624	851	1405	1179	921	1317	1131	1131	1	1	1h4	-
orb04	1	759	833	1325	1236	944	1327	1088	1119	1	1	2h6	-
orb05	1	630	801	1155	1152	812	1028	941	964	1	1	0h57	-
orb06	1	659	930	1330	1190	939	1189	1081	1081	1	1	0h37	-
orb07	1	286	345	475	504	366	465	422	433	1	1	0h6	-
orb08	1	585	894	1225	1107	894	1127	996	1051	1	1	0h16	-
orb09	1	661	705	1189	1262	894	1176	1031	1019	1	1	2h39	-
orb10	1	652	868	1303	1113	911	1131	1012	1012	1	1	1h11	-
swv01	1	1219	1366	2154	1737	1366	1877	1600	1571	1.372	2	1h0	1h0
swv02	1	1259	1475	2157	1706	1475	1981	1621	1643	1.376	1	0h29	-
swv03	1	1178	1328	2019	1806	1328	1755	1567	1572	1.411	2	1h29	1h29
swv04	1	1161	1366	2015	1874	1388	1888	1681	1661	1.462	1	1h5	1h5
swv05	1	1235	1411	2003	1922	1411	1956	1608	1608	1.472	1	0h35	0h35
swv06	1	1229	1477	2519	2140	1521	2275	1907	1905	1.939	1	1h52	1h52
swv07	1	1128	1394	2268	2146	1409	1960	1867	1884	1.901	2	3h13	3h13
swv08	1	1330	1586	2554	2231	1585	2196	2049	2058	2.017	1	2h9	2h9
swv09	1	1266	1594	2498	2247	1594	2218	1940	1910	1.977	2	1h23	1h23
swv10	1	1159	1560	2352	2337	1560	2276	1947	1974	2.063	2	1h50	1h50
ft06	2	86	92	101	130	93	104	96	96	1	1	1s	-
abz5	2	1736	1868	2188	2327	1868	2220	2056	2069	1.349	1	2h19	-
abz6	2	1376	1472	1726	1876	1472	1701	1547	1549	1	1	1h24	-
ft10	2	1262	1352	1812	1735	1397	1703	1620	1635	1	1	4h16	-
orb01	2	1286	1571	2015	2009	1580	1955	1855	1869	1	1	2h17	-
orb02	2	1342	1368	1664	1753	1389	1612	1510	1494	1	1	1h38	-
orb03	2	1248	1454	2040	1879	1485	1999	1736	1778	1	1	1h53	-
orb04	2	1518	1552	1965	2135	1552	1979	1750	1750	1	1	2h23	-
orb05	2	1260	1384	1817	1733	1384	1535	1481	1451	1	1	0h54	-
orb06	2	1318	1589	2049	1912	1589	1810	1722	1722	1	1	1h9	-
orb07	2	572	620	732	773	620	722	655	655	1	1	0h6	-
orb08	2	1170	1479	1898	1628	1480	1651	1588	1563	1	1	0h41	-
orb09	2	1322	1366	1738	1770	1387	1683	1624	1607	1	1	3h48	-
orb10	2	1304	1494	1857	1801	1503	1717	1662	1624	1	1	3h22	-
swv01	2	2438	2585	3385	2808	2585	3128	2849	2836	1.77	1	0h32	-
swv02	2	2518	2734	3346	3058	2733	3230	2850	2843	1.875	1	0h30	0h30
swv03	2	2356	2506	3195	2995	2505	3040	2738	2737	1.845	1	0h46	0h46
swv04	2	2322	2527	3217	3292	2529	3152	2866	2842	1.948	2	1h19	1h19
swv05	2	2470	2646	3237	3287	2645	3063	2786	2780	1.942	2	0h40	0h40
swv06	2	2458	2697	3579	3696	2696	3329	3160	3144	2.56	2	1h46	1h46
swv07	2	2256	2522	3384	3473	2522	3383	2977	2970	2.432	1	1h43	-
swv08	2	2660	2916	3757	3942	2916	3475	3292	3267	2.673	1	1h22	-
swv09	2	2532	2860	3767	3525	2858	3445	3100	3085	2.522	1	1h13	1h13
swv10	2	2318	2711	3977	3543	2710	3656	3194	3168	2.583	2	1h56	1h56
ft06	5	215	221	229	261	221	224	**221**	**221**	1	1	4s	-
abz5	5	4340	4472	4797	5278		4530	**4472**	**4472**	2.029	1	1h36	-
abz6	5	3440	3536	3693	4105		3538	**3536**	3537	1.558	1	2h24	-
ft10	5	3155	3238	3731	3492	3237	3569	3287	3307	1.407	1	1h4	1h4
orb01	5	3215	3500	3962	4391	3500	3850	3747	3799	1.545	1	1h3	-
orb02	5	3355	3355	3659	3768		3402	**3355**	**3355**	1.462	1	2h3	-
orb03	5	3120	3326	3949	4224	3326	3701	3488	3534	1.527	2	0h44	0h44
orb04	5	3795	3829	4132	4488	3829	3964	3885	3910	1.61	1	1h38	-
orb05	5	3150	3180	3594	3796		3309	**3180**	**3180**	1.402	1	1h0	-
orb06	5	3295	3566	4080	4201	3567	3739	3659	3679	1.597	1	1h1	1h1
orb07	5	1430	1450	1584	1669	1461	1508	1477	1477	1	1	0h36	-
orb08	5	2925	3234	3581	3505	3234	3341	3299	3315	1.336	1	1h19	-
orb09	5	3305	3349	3801	3757	3349	3478	3384	3391	1.472	1	1h28	-
orb10	5	3260	3450	3675	3889		3509	3450	3474	1.503	1	1h8	-
swv01	5	6095	6242	7042	6589	6242	6693	6407	6379	2.81	1	0h37	0h37
swv02	5	6295	6511	7123	7101	6510	6975	6622	6604	2.964	1	0h42	0h42
swv03	5	5890	6040	6724	6827	6040	6604	6130	6129	2.862	1	0h35	-
swv04	5	5805	6010	6636	7372	6009	6632	6186	6152	2.886	1	0h41	0h41
swv05	5	6175	6351	6942	7271	6350	6709	6490	6480	2.94	1	0h42	0h42
swv06	5	6145	6384	7288	8063	6386	6971	6669	6680	3.798	1	1h23	1h23
swv07	5	5640	5906	6702	7290	5906	6442	6206	6201	3.552	1	2h1	-
swv08	5	6650	6906	7633	8194	6906	7617	7117	7117	3.962	1	1h16	-
swv09	5	6330	6658	7519	7952	6655	7208	6827	6847	3.842	1	1h14	1h14
swv10	5	5795	6164	7099	7816	6164	7168	6708	6737	3.795	1	2h25	-
ft06	10	430	436	444	496	436	436	**436**	**436**	1	1	8s	-
abz5	10	8680	8812	9118	9857		8812	**8812**	**8812**	2.851	1	1h5	-
abz6	10	6880	6976	7105	7789		6976	**6976**	**6976**	2.202	1	1h28	-
ft10	10	6310	6393	6813	6644	6393	6550	6411	6407	1.98	1	0h51	-
orb01	10	6430	6715	7177	8280	6715	7117	6898	6956	2.119	2	1h0	1h0
orb02	10	6710	6710	6904	7155		6710	**6710**	**6710**	2.048	1	0h57	-
orb03	10	6240	6446	7038	7639	6446	6728	6583	6587	2.076	1	0h43	0h43
orb04	10	7590	7624	7927	8297	7624	7724	7647	7662	2.277	1	1h12	-
orb05	10	6300	6330	6637	7021		6330	6330	6330	1.942	1	0h49	-
orb06	10	6590	6861	7375	7938	6861	7043	6937	6987	2.188	1	0h56	-
orb07	10	2860	2880	3018	3109	2890	2923	2907	2908	1	1	1h8	-
orb08	10	5850	6159	6650	6486	6159	6262	6211	6186	1.854	2	1h9	1h9
orb09	10	6610	6654	7068	7017	6654	6758	6694	6683	2.052	1	1h24	-
orb10	10	6520	6710	6935	7239		6780	6729	6734	2.105	1	1h2	-
swv01	10	12190	12337	13137	12684	12334	12759	12451	12451	4.058	1	0h32	0h32
swv02	10	12590	12806	13418	13758	12806	13289	12942	12945	4.239	1	0h38	-
swv03	10	11780	11930	12614	13302	11930	12438	12026	12026	4.073	1	0h37	-
swv04	10	11610	11815	12330	14678	11815	12301	11859	11895	4.081	1	0h45	-
swv05	10	12350	12526	13117	13958	12526	12830	12609	12602	4.202	1	0h36	-
swv06	10	12290	12529	13398	15621	12529	13073	12756	12803	5.393	1	2h21	2h21
swv07	10	11280	11546	12310	13585	11546	12103	11739	11757	5.037	1	1h57	-
swv08	10	13300	13556	14376	15801	13556	14266	13813	13818	5.707	1	2h18	-
swv09	10	12660	12988	13849	15227	12988	13476	13188	13190	5.467	1	1h30	-
swv10	10	11590	11928	12968	15132	11926	12901	12433	12432	5.362	1	4h30	4h30

TABLE 1. Detailed solutions for fixed multiplicity instances

Name	$\hat{C}$ L.B	$\hat{C}^+$ L.B	FSA Solution	SPT Solution	LPSA L.B	$\alpha = 0$ Solution	H1 Solution	H2. Solution	Factor	Opt. Step	Time Step 1	Time Step 2
ft06	195	197	215	228	197	202	**197**	198	1	1	3s	-
abz5	4382	4395	5034	5320	4394	4891	4591	4593	2.081	2	2h56	1h7
abz6	3768	3768	4194	4447	3799	4031	3916	3915	1.665	2	3h28	3h1
ft10	3267	3350	3821	3703	3350	3571	3382	3382	1.452	1	0h58	-
orb01	3550	3835	4230	4305	3835	4264	4044	4047	1.6	1	1h4	0h50
orb02	3658	3680	4101	4395	3734	3888	3791	3777	1.552	1	2h44	0h0
orb03	3468	3674	4249	4432	3674	4056	3776	3797	1.587	1	1h6	-
orb04	3733	3767	4336	4223	3767	3911	3773	3773	1.628	1	1h45	-
orb05	3019	3163	3730	4170	3166	3558	3334	3331	1.43	2	1h17	1h9
orb06	3261	3532	3967	4340	3532	3809	3628	3628	1.574	1	1h13	-
orb07	1689	1709	1936	1974	1723	1756	1733	1727	1	1	0h41	-
orb08	2977	3286	3625	3572	3287	3486	3332	3333	1.35	2	0h52	0h22
orb09	3206	3250	3786	3849	3259	3487	3306	3320	1.478	1	1h57	1h54
orb10	3407	3575	3955	4622	3591	3778	3726	3720	1.563	2	4h11	2h0

TABLE 2. Detailed solutions for variable multiplicity instances $[(8,4,3,1,3,9)$ for ft06 and $(8,4,3,1,3,9,7,4,10,1)$ for the rest]

In the fourth column we present the fraction of the optimality gap that was closed when using our method. The comparison is with the best between FSA and SPT, where $\hat{C}$ is used as a lower bound. That is,

$$\text{Gap Reduction} = 1 - \frac{\min(H1, H2) - C(LPSA)}{\min(FSA, STT) - \hat{C}}.$$

Next we present the average optimality gap of LPSA,

$$\text{Gap} = \frac{\min(H1, H2) - C(LPSA)}{\min(H1, H2)}.$$

Finally the average CPU time required to solve MILP2 is presented in the rightmost column. The reported time also includes the time required for the second step (rounding down) for instances with factor $f > 1$.

It is apparent from Table 3 that LPSA is capable of producing better solutions than FSA and SPT for a large variety of problem instances with multiplicity of up to ten. In fact we got strictly better solutions for 109 out of the 110 test problems (with the exception of swv01 with multiplicity 2 for which SPT delivered the best solution).

It is apparent from Table 3 that LPSA is capable of producing better solutions than FSA and SPT for a large variety of problem instances with multiplicity of up to ten. In fact, we got strictly better solutions for 109 out of the 110 test problems, with the exception of swv01 with multiplicity 2 for which SPT delivered the best solution.

We also observed that the linear relaxation we employ delivers better lower bounds than the simple $\hat{C}$, and in most cases also better than $\hat{C}^+$. However, for larger problem instances, we could only get a slight increase of the lower bound. We believe that this is partially due to the effect of the larger factors that are used with these instances. With additional computational effort or more sophisticated methods, MILP2 could be solved without factorizing the processing times and tighter lower bounds could be obtained.

As mentioned, we tried to use factors that kept the size of the linear programs solved by Algorithm 1 approximately constant (except for the smallest instances that could be solved with their original processing times). Hence, relatively similar CPU times were obtained for all problem sizes and multiplicities.

In Table 4, we summarize our test of the effect of the number of α-points used on the best solutions found by the LPSA heuristics. The table presents the average optimality gap obtained when using 2,11,101,1001 α-points as well as single points with $\alpha = 0$. The reported result are the average of the best result between H1 and H2 for each of the tested problem multiplicity. It appears that while a significant improvement is obtained by applying the α-points methods the marginal improvement obtained from increasing the number of α is not dramatic. Increasing the number of α-points from 11 to 1001 resulted

Multiplicity / Dimensions	Solution Improvement	LB Improvement	Gap Reduction	Gap	CPU Time
1 - (24)	10.56%	24.42%	67.14%	14.76%	1h49m
6×6 (1)	9.23%	17.31%	68.18%	11.86%	1s
10×10 (13)	9.61%	26.01%	71.09%	13.29%	1h25m
20×10 (5)	11.10%	13.11%	64.55%	13.20%	1h24m
20×15 (5)	12.77%	20.31%	59.28%	20.74%	3h45m
2 - (24)	9.31%	9.67%	67.22%	9.22%	1h45m
6×6 (1)	4.95%	7.53%	80.00%	3.13%	1s
10×10 (13)	9.04%	10.46%	67.79%	9.19%	2h02m
20×10 (5)	8.08%	6.88%	64.68%	7.39%	1h04m
20×15 (5)	12.10%	10.83%	65.69%	12.35%	2h04m
5 - (24)	7.20%	3.77%	85.89	2.19%	1h24m
6×6 (1)	3.49%	2.71%	100%	0%	7s
10×10 (13)	8.60%	3.89%	90.40%	1.60%	1h28m
20×10 (5)	6.60%	2.87%	83.23%	1.86%	0h56m
20×15 (5)	7.49%	4.58%	74.01%	4.51%	1h59m
10 - (24)	3.91%	1.38%	84.4%	0.95%	1h23
6×6 (1)	1.80%	1.40%	100%	0%	8s
10×10 (13)	4.00%	2.00%	93.15%	0.62%	1h10m
20×10 (5)	3.56%	1.46%	86.59%	0.75%	0h41m
20×15 (5)	4.45%	2.29%	76.70%	2.17%	2h53m
Variable - (14)	8.71%	3.65%	85.93%	2.24%	2h29m
6×6 (1)	8.37%	1.02%	100%	0%	2s
10×10 (13)	8.74%	3.85%	84.85%	2.42%	2h40m
all (110)	8.60%	8.57%	78.33%	6.20%	1h42m

TABLE 3. Summary statistics of the numerical experiment

Multiplicity	$\alpha = 0$	2 α-points	11 α-points	101 α-points	1001 α-points
1	32.94%	29.31%	15.31%	14.96%	14.76%
2	20.47%	16.98%	10.15%	9.39%	9.22%
5	6.37%	4.01%	2.48%	2.23%	2.19%
10	2.77%	1.51%	1.04%	0.98%	0.95%
Variable	6.89%	3.81%	2.41%	2.24%	2.24%
all	14.52%	11.79%	8.60%	8.46%	8.03%

TABLE 4. Effect of the number of α-points on the optimality gap

in an average reduction of 0.5% of the gap. However, since the processing time of the LPSA heuristics is negligible, as compared to the substantial computational effort required for the solution of MILP2, we recommend using at least 1001 points.

Table 3 does not tell the whole story about the effect of the multiplicity on LPSA effectiveness. As is apparent in the table, the **relative** optimality gap diminishes as the multiplicity of the problem increases.

Multiplicity	LPSA	FSA
	Avg. NAOG	Avg. NAOG
1	100%	235%
2	101%	252%
5	47%	241%
10	39%	233%

TABLE 5. NAOG

This stems naturally from the asymptotic optimality property of LPSA, the approximation algorithm that underlines our heuristic, and it is also the case for other competing methods such as the FSA. However, our experiment supports the observation that LPSA delivers a smaller **absolute** optimality gap as the multiplicity of the problems increases.

Statistical analysis of absolute gaps over different problem instances is not straightforward, because the optimal makespan largely varies among the instances and the absolute gap tends to vary accordingly. To overcome this issue, we defined the notion of normalized absolute optimality gap (NAOG). NAOG is calculated by dividing the absolute optimality gap of instances with multiplicity greater than one, by the absolute gap of an instance of the same problem with multiplicity of one (i.e., with a single instance of each job). For example, the absolute optimality gap of LPSA for 'ft10' with multiplicity of one is $1065 - 859 = 206$ and with a multiplicity of five is $6450 - 6393 = 57$. Now, the NAOG of 'ft10' with a multiplicity of five is $57/206 = 23\%$. One can use these normalized values over different problem instances to understand the effects of multiplicity on this gap when comparing different algorithms.

In Table 5, we present the average NAOG of our 24 problem instances for multiplicities 1,2,5, and 10 for the solutions obtained from LPSA and FSA. The NAOG for both LPSA and FSA is calculated relative to the optimality gap of LPSA with multiplicity one. Hence, the NAOG obtained by LPSA for instances with multiplicity of one is by definition 100%. The results presented in the table implies that in LPSA heuristics, the effect of multiplicity on the absolute gap is notable for multiplicity of five and more.

We point out that the absolute, rather than the relative, optimality gap is of greater interest for the planner, because this value represents the actual amount of "money that may have been left on the table". The results presented in Table 5 show that when using LPSA, this amount declines quickly as the multiplicity increases.

7. Conclusions

In this study, we introduced a family of asymptotic approximation algorithms for the high multiplicity jobshop minimum makespan problem that are based on a solution of a linear programming relaxation of a time-indexed mixed integer linear program (MILP) model. The solution delivered from these algorithms is given in terms of a dispatching rule that is derived from the start, completion, or α-point times of the operations in the LP relaxation of the problem. We stress that a feasible solution of MILP3 that can be obtained in polynomial yields the same approximation guarantee as the optimal solution of MILP3 (see proof of Proposition 2.2). Therefore LPSA is a polynomial time approximation algorithm that dominates previously known methods in terms of the theoretic approximation guaranty.

LPSA-H1 and H2 are heuristics methods based on LPSA that are shown to deliver substantially better solutions than previously known heuristics for large set of OR-Lib test problems with moderate

multiplicity. In addition, apparently not only the relative optimality gap obtained from these methods decreases as the job multiplicity increases, but so does the absolute gap. This is a unique property of the method that is stronger than asymptotic optimality.

Naturally, to facilitate asymptotic approximation analysis, most of the high multiplicity scheduling literature assume a very large number of jobs in each class. This is also the case for the analysis of our approximation LPSA algorithm. However, in many applications, jobshop (as opposed to flow shops) is used to produce many classes of items at small quantities in a flexible production environment. Current technology and managerial trends advocate for agility and make-to-order policy rather than for mass production. For these applications, the fact that our heuristic method outperforms previous methods for problem instances with moderate multiplicity is significant.

Since LPSA and the proposed heuristics can be started with arbitrary initial inventories expressed by $s_{r,o}$, it is suitable for operation in a rolling horizon setting. That is, the algorithm is applicable for operation in an environment in which information on new orders, machine breakdowns, and yield issues are continually revealed.

Our heuristic methods are based on a solution of MILP2. We show that this MILP can be solved in pseudo-polynomial time. Indeed, finding the optimal solutions to our test instances came at a significant computational cost. However, we would like to point out that the techniques we used in our experiment for solving MILP2 are sufficient to tackle many real-life problems. Consider, for example, a detailed monthly plan for a jobshop with 15 machines. Assume that the month consists of 250 net working hours, the total number of operations types is $O = 225$ (say, 15 job classes with 15 operation per job), and assume that each operation takes one hour on average, $\bar{p}_{r,o} = 1$ hour. If the time is finely discretized into 5-minute periods, which is enough to represents the accuracy of the data in most applications, then the planning horizon is 3000 periods long and the average processing time per operation is 12 periods. $P = \bar{p}_{r,o} \times O = 2700$, and so the number of non zeros in LP1 is about $T(P+O) = 3000(2700+225) = 8,775,000$. We demonstrated that even larger instances, in terms of non zeros in the linear program, can be solved within a couple of hours without factorizing the processing times.

In addition, a substantial reduction in the run time could be obtained by a more careful implementation of the algorithm, e.g., by devising a columns generation procedure for LP1. Alternatively, LPSA can be based on completely different MILP formulations of the jobshop problems, possibly one that is not time-indexed and hence may be less sensitive to the digit size of the processing times.

We believe that this study paves the way for future interesting research. First, LPSA and LPSA-H can be generalized for similar jobshop problems, such as the minimum flow time $J/multi/\sum c_i$, e.g., Bertsimas et al. (2003), as well as more general models with holding costs that are specific to each stage in the production and with limitations on the buffer sizes. Furthermore, similar LP-based algorithms can be devised for other complex systems such as assembly systems, multi-stage production systems, scheduling of automated storage and retrieval systems (AS/RS), and the scheduling of railway systems, etc.

References

Adams, J., E. Balas, D. Zawack. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Sci.* **34**(3) 391–401.

Amin, J., M.A. Shafia, R. Tavakkoli-Moghaddam. 2011. A hybrid algorithm based on particle swarm optimization and simulated annealing for a periodic job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology* **54**(1-4) 309–322.

Beasley, J.E. 1990. Or-library, http://people.brunel.ac.uk/∼mastjjb/jeb/info.html. Web site.

Bertsimas, D., D. Gamarnik. 1999. Asymptotically optimal algorithm for job shop scheduling and packet routing. *J. of Algorithms* **33**(2) 296–318.

Bertsimas, D., D. Gamarnik, J. Sethuraman. 2003. From fluid relaxations to practical algorithms for job shop scheduling: The holding cost objective. *Oper. Res.* **51**(5) 798–813.

Bertsimas, D., J. Sethuraman. 2002. From fluid relaxations to practical algorithms for job shop scheduling: The makespan objective. *Math. Programming* **92**(1) 61–102.

Blackstone, J.H., D.T. Phillips, G.L. Hogg. 1982. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research* **20**(1) 27–45.

Boudoukh, T., M. Penn, G. Weiss. 2001. Scheduling job shop with some identical or similar jobs. *J. of Scheduling* **4** 177–199.

Brauner, N., Y. Crama, A. Grigoriev, J. van de Klundert. 2005. A framework for the complexity of high-multiplicity scheduling problems. *Journal of Combinatorial Optimization* **9**(3) 313–323.

Chekuri, C., S. Khanna. 2004. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis"*, chap. Approximation Algorithms for Minimizing Average Weighted Completion Time. CRC Press, Boca Raton, Florida 33431, 11–1–11–30.

Correa, J.R., M.R. Wagner. 2005. Lp-based online scheduling: From single to parallel machines. Michael Jnger, Volker Kaibel, eds., *Integer Programming and Combinatorial Optimization*, *Lecture Notes in Computer Science*, vol. 3509. Springer Berlin Heidelberg, 196–209.

Dai, J.G., G. Weiss. 1996. Stability and instability of fluid models for certain re-entrant lines. *Math. Oper. Res.* **21**(1) 115–134.

Dai, J.G., G. Weiss. 2002. A fluid heuristic for minimizing makespan in job-shops. *Oper. Res.* **50**(4) 692–707.

Goemans, M., M. Queyranne, A.S. Schulz, M. Skutella, Y. Wang. 2002. Single machine scheduling with release dates. *SIAM Journal on Discrete Mathematics* **15**(2) 165–192.

Goldberg, Leslie Ann, Mike Paterson, Aravind Srinivasan, Elizabeth Sweedyk. 1997. Better approximation guarantees for job-shop scheduling. *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '97, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 599–608.

Hall, N.G., T.E. Lee, M.E. Posner. 2002. The complexity of cyclic shop scheduling problems. *J. of Scheduling* **5**(4) 307–327.

Kechadi, M-Tahar, Kok Seng Low, G. Goncalves. 2013. Recurrent neural network approach for cyclic job shop scheduling problem. *Journal of Manufacturing Systems* **32**(4) 689 – 699.

Kimbrel, T., M. Sviridenko. 2008. High-multiplicity cyclic job shop scheduling. *Operations Research Letters* **36**(5) 574 – 578.

Lee, T.E., M.E. Posner. 1997. Performance measures and schedule patterns in periodic job shops. *Oper. Res.* **45**(1) 72–91.

Lenstra, J.K., A.H.G. Rinnooy-Kan. 1979. Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics* **4**(8) 121–140.

Leung, J.M.Y., G. Zhang, X. Yang, R. Mak, K. Lam. 2004. Optimal cyclic multi-hoist scheduling: A mixed integer programming approach. *Operations Research* **52**(6) 965976.

Nowicki, E., C. Smutnicki. 1996. A fast taboo search algorithm for the job shop problem. *Management Science* **42**(6) 797–813.

Savelsbergh, Martin W. P., R. N. Uma, Joel Wein. 1998. An experimental study of lp-based approximation algorithms for scheduling problems. *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '98, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 453–462.

Sevast'janov, S.V. 1994. On some geometric methods in scheduling theory: a survey. *Discrete Applied Mathematics* **55**(1) 59 – 82.

Shmoys, D.B., C.Stein, J.Wein. 1994. Improved approximation algorithms for shop scheduling problems. *SIAM Journal Computing* **23**(3) 617–632.

Skutella, M. 2006. List scheduling in order of α-points on a single machine. *Lecture Notes in Computer Science* **3484** 250291.

Sotskov, Y.N., N.V. Shakhlevich. 1995. Np-hardness of shop-scheduling problems with three jobs **59**(3) 237–266.

Weiss, G. 2008. A simplex based algorithm to solve separated continuous linear programs. *Mathematical Programming Series A* **115**(1) 151–198.

Williamson, D. P., L. A. Hall, J. A. Hoogeveen, C. A. J. Hurkens, J. K. Lenstra, S. V. Sevast'janov, D. B. Shmoys. 1997. Short shop schedules. *Operations Research* **45**(2) 288–294.