Linear programming assisted (not embedded) genetic algorithm for flexible jobshop scheduling with lot streaming

Published in 2018 in Computers & Industrial Engineering, Vol. 117, 319-335

Please cite this article as:

Defersha, F. M., and Bayat Movahed, S. (2018) Linear programming assisted (not embedded) genetic algorithm for flexible jobshop scheduling with lot streaming, Computers & Industrial Engineering, Vol. 117, pp.319-335

The online version can be found at the following link:

https://www.sciencedirect.com/science/article/pii/S0360835218300469

Linear programming assisted (not embedded) genetic algorithm for flexible jobshop scheduling with lot streaming

Fantahun M. Defersha*, Saber Bayat-Movahed

School of Engineering, University of Guelph, 50 Stone Road East, Guelph, Ontario, Canada, N1G 2W1

Abstract

The hybridization of metaheuristics with other techniques for optimization has been one of the most interesting trends. The focus of research on metaheuristics is also becoming problem oriented rather than algorithm oriented. This has led researchers to try combining different algorithmic components in order to design more powerful algorithms. In this paper, we developed a linear programming assisted genetic algorithm for solving a flexible jobshop lot streaming problem. The genetic algorithm searches over both discrete and continuous variables in the problem solution space. A linear programming is used to assist the genetic algorithm by further refining promising solutions in a population periodically through determining the optimal values of the continuous variables corresponding to those promising solutions. This is different from one common way of hybridization referred to as linear programming embedded metaheuristics where the algorithm searches only over the integer variables and a linear programming subproblem is solved corresponding to every solution visited, which can be computationally prohibitive. Numerical examples showed that the proposed linear programming assisted (not embedded) genetic algorithm is superior to the embedded approach and as well as to a resource intensive multi-population pure parallel genetic algorithm.

Keywords: Flexible Job-shop Scheduling; Lot Streaming; Hybrid Genetic Algorithm; Linear Programming Embedded; Linear Programming Assisted.

1. Introduction

Over the last decades, many efforts have been made by researchers to explore metaheuristics. This is due to the fact that metaheuristics have been proven to be very promising alternatives to classical optimization methods. However, in recent years, it has become apparent that pure applications of metaheuristics are limiting when solving complex and large problems. Hence, many researchers from different disciplines started to undertake research in exploring hybrid metaheuristics (HMH). For example, a conference series in integrating artificial intelligence and operation research techniques (CPAIOR Conferece series, n.d) has been held yearly in different countries since 2004. In that same year, a series of international workshops on HMH (HM 2004, 2005, ..., 2016, ...) was launched and its 10th workshop was held in 2016 in Plymouth, United Kingdon (Blesa *et al.*, 2016). A new artificial term "matheuristics" was introduced in another workshop series, a *Workshop on Mathematical Contribution to Metaheuristics* (Matheuristics2006, 2006), and subsequently used by many researchers to refer to optimization algorithms made by the interoperation of (meta)heuristics and mathematical programming (MP) techniques. Books that are specifically devoted to HMH

*Corresponding author

Email address: fdefersh@uoguelph.ca (Fantahun M. Defersha)

have been published (e.g. Blum *et al.* (2008); Maniezzo *et al.* (2009); Talbi (2013)). This growing interest in HMH is an indication of their suitability in solving complex problems. The questions regarding the proper integration of different algorithmic components and the adequate analysis of results are becoming contemporary and attracting a large number of researchers. As a result, articles documenting the successful applications of HMH and their design and implementation issues are flourishing in large numbers. Thus providing a comprehensive review of those articles is beyond the scope of this paper. Instead, in the following sections we briefly review articles that deal with the classification of HMH and the hybridization of metaheuristics with linear programming as they are relevant to the work presented in this paper. Interested readers can find a comprehensive survey on HMH in Blum *et al.* (2011).

1.1. Classification of hybrid metaheuristics

Hybrid-metaheuristics (HMH) can be classified in many different ways. Talbi (2002) provided a hierarchical classification based on the design issues in hybridization. At the top of the hierarchy, Talbi classified HMHs as low-level versus high-level hybridizations. In low-level hybridization, a functional component of an optimization algorithm is exchanged with other metaheuristic. In high-level hybridization, on the other hand, apart from a controlled exchange of information, there is no internal relation between the individual optimization algorithms and all the algorithms preserve their identities. Within the low- and high-levels classifications of HMHs, Talbi further distinguished the hybridization of algorithms as teamwork versus relay. In teamwork hybridizations, optimization algorithms cooperate with each other so that each algorithm performs a search in the solution space simultaneously, whereas relay hybridizations are those in which the output of an optimization algorithm is used as an input for the other algorithm. Raidl (2006) provides a more detail classification (see Figure 1) of HMHs based on four attributes: (1) the types of algorithms which may be combined, (2) level of hybridization, (3) order of execution, and (4) control strategy. According to the first attribute, metaheuristics may be hybridized with other metaheuristics, problem-specific algorithms, simulations, exact techniques, heuristics, and soft computing methods. The classifications based on the 2nd and 3rd attributes are more elaborated versions of the hierarchical classification in Talbi (2002) discussed previously. Based on control strategies, fourth attribute, hybrid metaheuristics are divided into integrative and collaborative categories. In the integrative approach, an optimization algorithm is embedded in a primary algorithm to work as a subordinate algorithm. However, in the collaborative approach, the relation between the optimization algorithms is limited to exchanging the information and the algorithms work separately. Other research articles also exist that provide classifications of HMH. Cotta *et al.* (2005) provide classification of parallel HMH. A classification and review of HMH that hybridize metaheuristics with exact method can be found in Puchinger and Raidl (2005). The exact methods hybridized with metaheuristic include branch and bound, dynamic programming, constraint programming, integer programming and linear programming (LP).

1.2. Hybridization of metaheuristics with LP

Consider a general mixed integer linear programming (MILP) mathematical model in Eqs. 1-6 where x and y are n_1-dimensional *continuous* and n_2-dimensional *integer* column vectors of variables, respectively. The entries for the parameters are $p \in \mathbb{R}^{n_1}$, $q \in \mathbb{R}^{n_2}$, $A \in \mathbb{R}^{m_1 \times n_1}$, $B \in \mathbb{R}^{m_1 \times n_2}$, $e \in \mathbb{R}^{m_1}$, $C \in \mathbb{R}^{m_2 \times n_2}$ and $f \in \mathbb{R}^{m_2}$. The constraint in Eq. (2) provides m_1 inequalities where each inequality is composed of both the continuous and the integer variables. The constraint in Eq. (3) represents m_2 inequalities where each inequality is involving of only the integer variables.

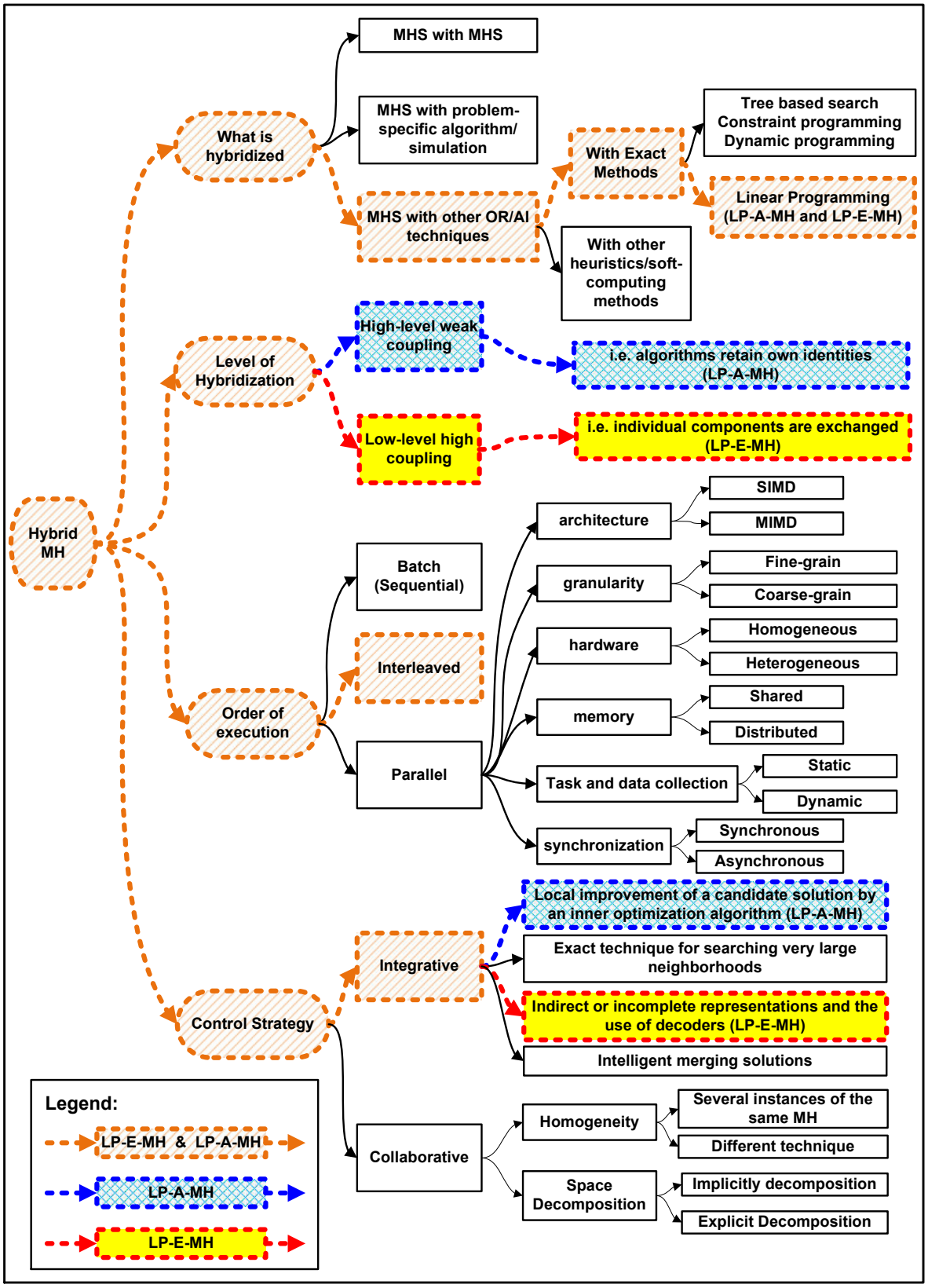


Figure 1: Classification of hybrid metaheuristic based on Raidl (2006) - *Within this general classification, we indicated the places of LP-E-MH and LP-A-MH (See Section 1.3)*

MILP:

$$\begin{aligned} \text{Minimize } Z &= p^T x + q^T y && (1) \\ \text{Subject to:} & \\ Ax + By &\leq e && (2) \\ Cy &\leq f && (3) \\ x &\in \mathbb{R}^{n_1} && (4) \\ y &\in \mathbb{Z}^{n_2} && (5) \\ x, y &\geq 0 && (6) \end{aligned}$$

A seemingly straightforward approach to solve the above model with a hybrid metaheuristic is to split the model into integer and continuous variable parts. One can then apply a metaheuristic to optimize the integer part only; before evaluating a solution, a linear programming solver is applied on an LP-subproblem in Eqs. 7-10 to augment the integer part with an optimal choice of continuous variables. To the best of our knowledge, this approach was first reported in Teghem *et al.* (1995). The authors applied this technique to solve a MILP model using a LP embedded simulated annealing (SA). Other early attempts of this approach are described in conjunction with GRASP by Neto and Pedroso (2003) and in conjunction with tabu search (TS) by Pedroso (2005). Similar strategies in solving MILP models were developed recently in Defersha and Chen (2008), Cao *et al.* (2009), Defersha and Chen (2009b), Defersha and Chen (2010a), Luo *et al.* (2014) and Shafigh *et al.* (2016). Mathematical models that consist of only binary and general integer were also solved in Rezazadeh *et al.* (2011) and Bayram and Şahin (2016) by LP-embedded metaheuristics. In those hybridizations, the metaheuristics search over the binary variables and for each solution visited a linear programming subproblem is formulated by relaxing the general integer variables. A complete solution is obtained by rounding the variables in the optimal solution of the LP-subproblem to the nearest integer values. Linear programming embedded probabilistic TS for facility layout was proposed in Kulturel-Konak (2012) where the authors did not formally develop MILP model. Instead, the solution representation of the TS is used as a modelling tool to decide the relative locations of departments. Given such a candidate solution, the actual locations and sizes of the departments are determined by solving a LP-subproblem.

LP-subproblem:

$$\begin{aligned} \text{Given } \{y | Cy &\leq f\} \\ \text{Minimize } Z &= p^T x && (7) \\ \text{Subject to:} & \\ Ax &\leq e - By && (8) \\ x &\in \mathbb{R}^{n_1} && (9) \\ x &\geq 0 && (10) \end{aligned}$$

Other LP-embedded metaheuristics, that are closely related but do not necessarily fall into the scheme of the MILP solution approach discussed above, have also been developed in literature. For example, a three-level LP-embedded genetic algorithm (GA) was developed in Urdaneta *et al.* (1999) for reactive power planning problem. A particular GA solution provides locations of reactive power sources, a LP-subproblem determines the magnitude of the power sources and finally a simulation model provides fitness measure for the overall solution. Hybrid GAs for water distribution system

optimization were developed in Reis *et al.* (2005) and Cisty (2010) where, for each solution visited by the GAs, more than one (perhaps many) LP-subproblems need to be solved. There are also many other hybridizations of metaheuristics with (integer) linear programming that completely fall outside the scheme of the hybridization discussed in this section. A review of many of those methods can be found in Raidl and Puchinger (2008).

1.3. LP-embedded vs LP-assisted metaheuristics

In this paper, we distinguished LP-embedded and LP-assisted metaheuristics (LP-E-MH and LP-A-MH, respectively) as two levels of hybridizations in solving MILP models following the strategy discussed in the previous section. In LP-E-MH, the solution representation of the metaheuristic involves only the integer variables. Hence, for each individual solution visited by the metaheuristic, a LP-subproblem has to be solved to augment this solution. Raidl (2006) categorized such approaches as decoder-based metaheuristics, in which a master algorithm acts on an implicit or incomplete representation of candidate solutions and a decoder is used to obtain corresponding actual solutions. According to Raidl, such a decoder can be virtually any kind of algorithm ranging from a simple problem specific heuristic to sophisticated exact optimization techniques or other OR/AI methods. All the hybrid metaheuristics reviewed in the previous section fall into this category where the decoder is a LP solver. However, a very large number of repeated calls to a LP solver in LP-E-MH approach can be computationally challenging when one attempts to solve a complex problem. To alleviate this problem, we proposed LP-A-MH as an alternative where the solution representation of the metaheuristic should include both the integer and the continues variables. In this way, the metaheuristics will have the capability to search the complete solution space without the LP solver. Instead, the LP solver will be used to assist the metaheuristic by further refining only promising solutions periodically after a certain number of iterations through determining the optimal values of the continuous variables corresponding to those promising solutions. In this case, the LP solver plays a role of a local improvement algorithm not as a sole decoder. The places of LP-E-MH and LP-A-MH are indicated in the classification of HMH in Figure 1.

1.4. The problem considered - lot streaming in flexible jobshop

A flexible jobshop scheduling problem (FJSP) consists of scheduling a given number of jobs on a given number of machines where a job is a batch of identical parts to be processed by following a given sequence of operations. In FJSP, unlike classical jobshop scheduling problem (JSP), operations can have alternative routes (machines). This makes FJSP a more complex problem than JSP due to the fact that in addition to operations assignment and sequencing, selection of a machine among the possible alternatives for each operation should also be accomplished simultaneously. Various meta-heuristics approaches such as TS, SA, and GA have been utilized to solve FJSP problem in literature. Among these approaches, the GA has been broadly applied (See for example Chen *et al.* (1999); Kacem (2003); and Zhang and Gen (2005)). Another extension to jobshop scheduling is the concept of lot streaming. The term lot streaming was first introduced by Reiter (1966) with the context of jobshop scheduling. It is a manufacturing technique in which jobs (or lots) are split into sublots in order to benefit from simultaneous processing of different operations of a job (Potts and Baker, 1989). In today's era of time-based competition (TBC), this process has been implemented by many top-notch companies in order to reduce their manufacturing lead time and improve their customer service (Blackburn, 1991; Bockerstette and Shell, 1993; Chang and Chiu, 2005). A very limited number of researches in jobshop scheduling with lot streaming (JSP-LS) have been reported in literature. In Dauzere-Peres and Lasserre (1997), an iterative approach for solving JSP-LS is developed. This approach solves JSP-LS, first with given lot sizes and then with given job sequences repeatedly until convergence is reached. Later, in Chan *et al.* (2004), a procedure using GA for

solving equal-sized lot streaming in JSP is introduced. Also, Chan *et al.* (2008b) proposed a method based on GA and simple dispatching rule to solve assembly jobshop scheduling problems with lot streaming. Likewise, Chan *et al.* (2008a), developed an approach based on the GA. In this approach, lot sizing and job-shop problems are solved simultaneously. In Defersha and Chen (2012), a more comprehensive mode for FJSP lot streaming (FJSP-LS) is proposed considering (1) unequal lot sizes, (2) sequence-dependent set-up time, (3) attached/detached set ups, (4) machine release dates, and (5) lag time. The authors developed a pure parallel GA that uses multiple population over multiple concurrently available computers to solve this comprehensive model. The model and the parallel genetic algorithm in Defersha and Chen (2012) is an extension of Defersha and Chen (2010b) where lot streaming was not considered. In this paper, we develop a sequential LP-assisted GA (LP-A-GA) that utilizes a single computational resource and still outperforms or equally performs as the resource intensive parallel pure GA presented in Defersha and Chen (2012). The remainder of this paper is organized as follows. In Section 2, a mixed integer-linear programming (MILP) model for FJSP-LS is represented. In Section 3, the common and distinct features of pure GA and the proposed hybrid GA are presented in details. Numerical examples are in Section 4. Finally, in Section 5, discussion and conclusions are provided.

2. Mathematical formulation

The main objective of this paper is to present a linear programming assisted GA to solve a scheduling problem (with lot streaming) in flexible job shop presented in Defersha and Chen (2012). However, for a better comprehension of this paper, we describe the problem and present the mathematical model here again.

2.1. Problem description and notations

Consider a job-shop consisting of M machines where machines with common functionalities are grouped into a department (e.g. turning machines in a turning department). Assume that the system is currently processing jobs from previous schedules and each machine m (where $m = 1, \ \ldots, \ M$) has a release date D_m at which time it will be available for next schedule. Consider also a total number of J independent jobs to be scheduled next in the system where a job is a batch of identical parts. The number of parts in a batch of job j (where $j = 1, \ \ldots, \ J$) is given by B_j and this batch is to be split into S_j number of unequal sublots (transfer batches). A decision variable $b_{s,j}$ is used to denote the size of sublot s (where $s = 1, \ \ldots, \ S_j$) of job j. Each sublot of job j is to undergo O_j number of operations in a fixed sequence such that each operation o (where $o = 1, \ \ldots, \ O_j$) can be processed by one of several eligible machines. $T_{o,j,m}$ is unit processing time for an operation o of a sublot of job j on machine m. An operation o of a sublot of job j can be started on an eligible machine m after lag time $L_{o,j}$ and after the setup is performed. The lag time $L_{o,j}$ is a waiting time that may be required either for cooling, drying or for some other purpose. The setup time for an operation o of job type j on machine m depends on the preceding operations and is denoted by $S_{o,j,m,o',j'}$, where operation o' of a sublot of job j' is the preceding operation on machine m. If operation o of sublot s of job j is the first operation to be processed on machine m, the setup time is simply represented as $S^*_{o,j,m}$. The setup time $S_{o,j,m,o',j'}$ (or $S^*_{o,j,m}$) for operation o of a sublot of job j can be overlapped with the processing time of operation $o-1$ of the same sublot if it is a detached setup and machine m is available for setup. The problem is to determine the size of each sublot, to assign the operation of each sublot to one of the eligible machines and to determine the sequence and starting time of the assigned operations on each machine. The objective is to minimize the makespan of the schedule. We next introduce some additional notations and then present a mixed integer linear programming (MILP) formulation for FJSP-LS.

Additional Parameters:

R_m Maximum number of production runs of machine m where production runs are indexed by r or $u = 1, 2, ..., R_m$; Each of these production runs can be assigned to at most one operation of one sublot. Thus the assignment of the operations to production runs of a given machine determines the sequence of the operations on that machine;

$P_{o,j,m}$ A binary data equal to 1 if operation o of a sublot job j can be processed on machine m, 0 otherwise;

$A_{o,j}$ A binary data equal to 1 if setup of operation o of a sublot of job j is attached (non-anticipatory), or 0 if this setup is detached (anticipatory);

Ω Large positive number.

Variables:

Continuous Variables:

c_{max} Makespan of the schedule

$c_{o,s,j,m}$ Completion time of operation o of sublot s of job j on machine m;

$\widehat{c}_{r,m}$ Completion time of the r^{th} run of machine m;

$b_{s,j}$ Size of sublot s of job j

Binary Integer Variables:

$x_{r,m,o,s,j}$ A binary variable which takes the value 1 if the r^{th} run on machine m is for operation o of sublot s of job j, 0 otherwise;

$y_{r,m,o,j}$ A binary variable which takes the value 1 if the r^{th} run on machine m is for operation o of any one of the sublots of job j, 0 otherwise;

$\gamma_{s,j}$ A binary variable that takes the value 1 if sublot s of job j is non-zero ($b_{s,j} \geq 1$), 0 otherwise,

$z_{r,m}$ A binary variable that takes the value 1 if the r^{th} potential run of machine m has been assigned to an operation, 0 otherwise;

2.2. MILP model for FJSP-LS

Following the problem description and using the notations given above, the MILP mathematical model for the FJSP-LS is presented below.

Minimize:

$$Objective = c_{max} \tag{11}$$

Subject to:

$$c_{max} \geq c_{o,s,j,m} \ ; \quad \forall (o, s, j, m) \tag{12}$$

$$\widehat{c}_{r,m} \geq c_{o,s,j,m} + \Omega \cdot x_{r,m,o,s,j} - \Omega \ ; \quad \forall(r,m,o,s,j) \tag{13}$$

$$\widehat{c}_{r,m} \leq c_{o,s,j,m} - \Omega \cdot x_{r,m,o,s,j} + \Omega \ ; \quad \forall(r,m,o,s,j) \tag{14}$$

$$\widehat{c}_{1,m} - b_{s,j} \cdot T_{o,j,m} - S^*_{o,j,m} - \Omega \cdot x_{1,m,o,s,j} + \Omega \geq D_m \ ; \quad \forall(m,o,s,j) \tag{15}$$

$$\widehat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m,o',j'} - \Omega \cdot (y_{r-1,m,o',j'} + x_{r,m,o,s,j}) + 2\Omega \geq \widehat{c}_{r-1,m} \ ; \\ \forall(r,m,o,s,j,o',j') | (r > 1) \tag{16}$$

$$\widehat{c}_{1,m} - b_{s,j} \cdot T_{o,j,m} - S^*_{o,j,m} \cdot A_{o,j} - \Omega \cdot (x_{1,m,o,s,j} + x_{r',m',o-1,s,j}) + 2\Omega \geq \widehat{c}_{r',m'} + L_{o,j} \ ; \\ \forall(m,r',m',o,s,j) | \{ \big((1,m) \neq (r',m') \big) \wedge (o > 1) \} \tag{17}$$

$$\widehat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m,o',j'} \cdot A_{o,j} - \Omega \cdot (y_{r-1,m,o',j'} + x_{r,m,o,s,j} + x_{r',m',o-1,s,j}) + 3\Omega \geq \widehat{c}_{r',m'} + L_{o,j} \ ; \\ \forall(r,m,r',m',o,s,j,o',j') | \{ (r > 1) \wedge (o > 1) \wedge (r,m) \neq (r',m') \wedge (o,j) \neq (o',j') \} \tag{18}$$

$$y_{r,m,o,j} \leq P_{o,j,m} \ ; \quad \forall(r,m,o,j) \tag{19}$$

$$y_{r,m,o,j} = \sum_{s=1}^{S_j} x_{r,m,o,s,j} \ ; \quad \forall(r,m,o,j) \tag{20}$$

$$\sum_{m=1}^{M} \sum_{r=1}^{R_m} x_{r,m,o,s,j} = \gamma_{s,j} \ ; \quad \forall(o,s,j) \tag{21}$$

$$b_{s,j} \leq B_j \cdot \gamma_{s,j} \ ; \quad \forall(s,j) \tag{22}$$

$$\gamma_{s,j} \leq b_{s,j} \ ; \quad \forall(s,j) \tag{23}$$

$$\sum_{s=1}^{S_j} b_{s,j} = B_j \ ; \quad \forall(j) \tag{24}$$

$$\sum_{j=1}^{J} \sum_{s=1}^{S_j} \sum_{o=1}^{O_j} x_{r,m,o,s,j} = z_{r,m} \ ; \quad \forall(r,m) \tag{25}$$

$$z_{r+1,m} \leq z_{r,m} \ ; \quad \forall(r,m) \tag{26}$$

$$x_{r',m,o',s,j} \leq 1 - x_{r,m,o,s,j} \ ; \quad \forall(r,r',m,o,o',s,j) | \{ (o' > o) \wedge (r' < r) \} \tag{27}$$

$$x_{r',m,o',s,j} \leq 1 - x_{r,m,o,s,j} \ ; \quad \forall (r, r', m, o, o', s, j) | \{(o' < o) \wedge (r' > r)\} \tag{28}$$

$$x_{r,m,o,s,j},\ y_{r,m,o,j},\ \gamma_{s,j} \text{ and } z_{r,m} \text{ are binary} \tag{29}$$

The complete description and the meanings of the objective function in Eq. (11) and the constraints in Eqs. (12)-(29) can be found in Defersha and Chen (2012). (Please note the purpose of this paper is to present an improved linear programming assisted genetic algorithm to solve this mathematical model).

3. Improved solution procedure

An initial parallel pure GA was developed in Defersha and Chen (2009a) and latter expanded in Defersha and Chen (2012) to solve the model presented in the previous section. In this section, we present a way of combining the pure GA with linear programming to create an efficient sequential hybrid algorithm. As noted in Section 1.3, we distinguished two levels of hybridization of metaheuristics with LP. In reference to GA, we refer to these levels of hybridization as LP-Assisted GA (LP-A-GA) and LP-Embedded GA (LP-E-GA). The LP-A-GA, the subject of this paper, while utilizes a single computational resource, outperforms or performs equally as the resource-intensive parallel pure GA. The following subsections convey the common and distinct features of both the pure and the hybrid GA.

3.1. Solution representation

The solution representations of LP-A-GA and LP-E-GA are adopted from that of the pure genetic algorithm developed in Defersha and Chen (2012). Hence, for better understanding of this paper, it is necessary to briefly describe the solution representation of the pure GA developed in Defersha and Chen (2012) to solve the FJSP-LS problem. As a way to portraying this representation, Defersha and Chen (2012) considered a small flexible job shop which processes three jobs with four machines. The data for this small system are given in Table 1. As per Defersha and Chen (2012), a possible operation assignment and sequencing for this small problem can be encoded as shown in Figure 2. In this representation, every sublot is considered as a job, and each gene in the chromosome is represented by a quadruple ($j,\ s,\ o,\ m$) denoting the assignment of the o^{th} operation of sublot s of job j to machine m. The sequence of the genes in the chromosome provides the sequences of the operations on every machine. For instance, through looking to the genes from left to right the assignment and sequencing of operations on machine-1 can be interpreted as follows: $(j1, s3, o1) \rightarrow (j3, s2, o3) \rightarrow (j3, s3, o3)$. These data are obtained from the genes at locations 10, 22 and 23 on the chromosome where $m = 1$. The assignment of operations to the other machines and their sequences as decoded from the chromosome is given in Table 2. In this solution representation, in order to ensure that precedence requirement of the operations of a particular sublot are not violated, for a given j and s, the gene (j, s, o, m) always lies to the right hand side of all the other genes (j, s, o', m') having $o' < o$. For detail discussion about this solution representation, one can refer to Defersha and Chen (2012).

In order to solve the proposed FJSP-LS model using GA, it is essential to include the number of sublots for each job and their sizes into the solution representation (Defersha and Chen, 2012). The chromosome in Figure 2 is capable of encoding only the assignment and sequencing of the operations of the sublots. Therefore, a left hand side segment (LHS-Segment) has been added to this chromosome as depicted in Figure 3. In this segment, every gene is represented by $\alpha_{s,j}$, which takes a random value in the interval [0, 1]. The value each $\alpha_{s,j}$ takes is used in Eq. (30) in order to compute the size of the s^{th} sublot of job j. It is possible for a certain sublot to have a size of zero

Table 1: An example small flexible job-shop problem (adopted from Defersha and Chen (2012))

Job	No. of Operations	Max No. of Sublots	Set of eligible machines for operation		
			$o1$	$o2$	$o3$
$j1$	3	3	$\{m1, m2\}$	$\{m3\}$	$\{m2, m4\}$
$j2$	2	2	$\{m3, m4\}$	$\{m2\}$	
$j3$	3	3	$\{m3\}$	$\{m2, m4\}$	$\{m1, m3\}$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
3, 2, 1, 3	**1, 2, 1, 2**	**2, 1, 1, 3**	**3, 1, 1, 3**	**2, 2, 1, 4**	**3, 2, 2, 4**	**3, 3, 1, 3**	**1, 3, 1, 1**	**3, 1, 2, 2**	**1, 1, 1, 2**	**3, 3, 2, 2**	**1, 2, 2, 3**	**1, 1, 2, 3**	**3, 1, 3, 3**	**1, 3, 2, 3**	**2, 1, 2, 2**	**1, 1, 3, 4**	**2, 2, 2, 2**	**1, 3, 3, 4**	**3, 2, 3, 1**	**3, 3, 3, 1**	**1, 2, 3, 2**	***j, s, o, m***

j **= job index,** ***s*** **= sublot index,** ***o*** **= operations index,** ***m*** **= machine index**

Figure 2: Representation of the assignment of operations to machines and their sequencing (adopted from Defersha and Chen (2012))

Table 2: Operation assignment and sequencing decoded from Figure 2 (adopted from Defersha and Chen (2012))

Machine	Operation assigned to production run							
	$r1$	$r2$	$r3$	$r4$	$r5$	$r6$	$r7$	$r8$
$m1$	$(j1, s3, o1)$	$(j3, s2, o3)$	$(j3, s3, o3)$					
$m2$	$(j1, s2, o1)$	$(j3, s1, o2)$	$(j1, s1, o1)$	$(j3, s3, o2)$	$(j2, s1, o2)$	$(j2, s2, o2)$	$(j1, s2, o3)$	
$m3$	$(j3, s2, o1)$	$(j2, s1, o1)$	$(j3, s1, o1)$	$(j3, s3, o1)$	$(j1, s2, o2)$	$(j1, s1, o2)$	$(j3, s1, o3)$	$(j1, s3, o2)$
$m4$	$(j2, s2, o1)$	$(j3, s2, o2)$	$(j1, s1, o3)$	$(j1, s3, o3)$				

if its corresponding $\alpha_{s,j}$ has a value equal to zero. If all $\alpha_{s,j}$ values are zero, the size of a sublot is computed by dividing the number of parts in a batch of job j (B_j) to the maximum number of sublots of S_j of that job. Thus, the maximum and actual numbers of sublots for each job and their sizes are encoded in the LHS-Segment. The solution representation discussed above is applicable both in the pure GA and LP-A-GA. However, in the case of LP-E-GA, the sizes of the sublots are always determined by solving a linear programming subproblem. Hence, the LHS-segment is used only to determine the number of sublots of each job. In that case, the gene $\alpha_{s,j}$ in the LHS-segment takes a binary value either 0 or 1 denoting whether sublot s of job j is created or not. During initialization and evolution in LP-E-GA, for each job j in the LHS-segment, it is necessary to have at least one of $\alpha_{s,j} = 1$ to ensure at least one sublot is created and the job is processed.

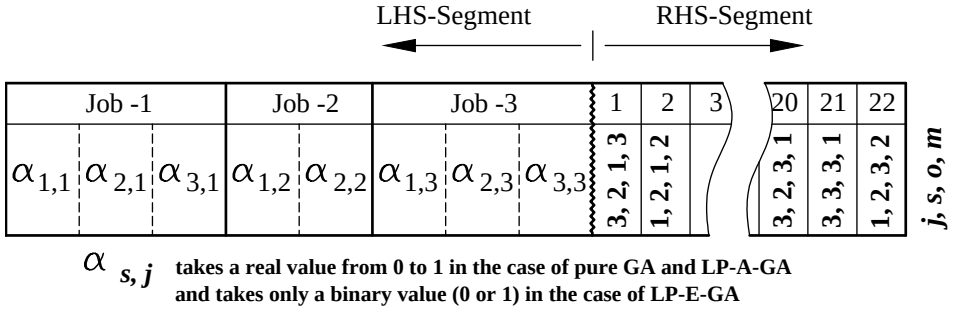


Figure 3: Solution representation (adopted from Defersha and Chen (2012))

$$b_{s,j} = \begin{cases} \frac{\alpha_{s,j}}{\sum_{s=1}^{S_j} \alpha_{s,j}} \times B_j & ; \text{if} \quad \sum_{s=1}^{S_j} \alpha_{s,j} > 0 \\ B_j / S_j & ; \text{otherwise} \end{cases} \tag{30}$$

3.2. Evaluation Procedure

The evaluation procedure of a solution of the pure GA can be found in Defersha and Chen (2012). This procedure utilizes the operations assignment and sequencing and sublot sizes decoded from a solution in order to determine the makespan in a step by step procedure. The LP-A-GA also applies this evaluation procedure with the exception that every certain number of generation, it applies a linear programming sub-problem to further refine promising solution. The LP-E-GA, on the other hand, solves LP-subproblem to evaluate each solution in every generation which makes the evaluation procedure computationally prohibitive.

3.3. Linear programming subproblem

A particular solution (chromosome) of the GA can be decoded to provide the values of the integer variables $x_{r,m,o,s,j}$, $y_{r,m,o,j}$, $\gamma_{s,j}$ and $z_{r,m}$ (and the values of the continuous variables which may not optimally correspond to these integer variable). In order to determine the values of the continuous variable that optimally correspond to the integer variable, a linear programming (LP) can be formulated. In formulation of this LP model, the constraints of the MILP model that are composed of only the integer variable are omitted. The rest of the equations in MILP model are modified slightly and inserted into LP subproblem. The object function remains unchanged and is given in Eq. (31) in the LP subproblem. Eq. (12) in the MILP is in Eq. (32) of the LP. In the MILP model, constraints in Eqs. (13) and (14) were used to set $c_{r,m} = c_{o,s,j,m}$ if variable $x_{r,m,o,s,j} = 1$. Nevertheless, these two equations can be merged together into Eq. (33) in the LP model since

$x_{r,m,o,s,j}$ is known. Knowing the values of $x_{r,m,o,s,j}$, Eq. (15) is replaced by Eq. (34). Similarly, Eqs. (16), (17) and (18) are replaced by Eqs. (35), (36) and (37), respectively. The constraints in Eq. (24) of the main MILP is also applicable in the LP-subproblem model and it is repeated in Eq. (38).

LP: given $(x_{r,m,o,s,j},\ y_{r,m,o,j},\ \gamma_{s,j},\ z_{r,m})$ **for all** (r, m, o, s, j)
Minimize:

$$Objective = c_{max} \tag{31}$$

Subject to:

$$c_{max} \geq c_{o,s,j,m}\ ;$$
$$\forall (o, s, j, m) | (\gamma_{s,j} = 1) \tag{32}$$

$$\widehat{c}_{r,m} = c_{o,s,j,m};$$
$$\forall (r, m, o, s, j) | \{(x_{r,m,o,s,j} = 1) \wedge (\gamma_{s,j} = 1)\} \tag{33}$$

$$\widehat{c}_{1,m} - b_{s,j} \cdot T_{o,j,m} - S^*_{o,j,m} \geq D_m\ ;$$
$$\forall (m, o, s, j) | \{(x_{1,m,o,s,j} = 1) \wedge (\gamma_{s,j} = 1)\} \tag{34}$$

$$\widehat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m,o',j'} \geq \widehat{c}_{r-1,m}\ ;$$
$$\forall (r, m, o, s, j, o', j') | \{(r > 1) \wedge (y_{r-1,m,o',j'} + x_{r,m,o,s,j} = 2) \wedge (\gamma_{s,j} = 1)\} \tag{35}$$

$$\widehat{c}_{1,m} - b_{s,j} \cdot T_{o,j,m} - S^*_{o,j,m} \cdot A_{o,j} - \geq \widehat{c}_{r',m'} + L_{o,j}\ ;$$
$$\forall (m, r', m', o, s, j) | \{[(1, m) \neq (r', m')] \wedge (o > 1) \wedge (x_{1,m,o,s,j} + x_{r',m',o-1,s,j} = 2)$$
$$\wedge (\gamma_{s,j} = 1)\} \tag{36}$$

$$\widehat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m,o',j'} \cdot A_{o,j} - \geq \widehat{c}_{r',m'} + L_{o,j}\ ;$$
$$\forall (r, m, r', m', o, s, j, o', j') | \{(r > 1) \wedge (o > 1) \wedge [(r, m) \neq (r', m')] \wedge [(o, j) \neq (o', j')]$$
$$\wedge (y_{r-1,m,o',j'} + x_{r,m,o,s,j} + x_{r',m',o-1,s,j} = 3) \wedge (\gamma_{s,j} = 1)\} \tag{37}$$

$$b_{s,j} \leq B_j \cdot \gamma_{s,j}\ ;\ \ \forall (s, j) \tag{38}$$

$$\sum_{\forall s | (\gamma_{s,j} = 1)} b_{s,j} = B_j\ ;\ \ \forall (j) \tag{39}$$

Once the LP-subproblem is solved, the optimal values of the continuous variables c_{max}, $c_{o,s,j,m}$, $\widehat{c}_{r,m}$ and $b_{s,j}$ are known for the chromosome under consideration. The fitness value of this chromosome is then set to c_{max}. The optimal value of each $b_{s,j}$ is also encoded back into the LHS-Segment of this chromosome by updating its $\alpha_{s,j}$ with $b_{s,j}/B_j$ for all (j, s).

3.4. LP-subproblem implementation approaches

In the LP-A-GA, improving a particular solution using a LP-subproblem involves: (i) adding the objective function and the constraints of the subproblem into a modelling environment, and (ii) solving it using a linear programming solver. These two steps are performed for a very large number of solutions during the entire search process. At first, one may think that a large number of calls to the linear programming solver of the second step can pose a major computational difficulty. However, a closer look into these two steps in a related study by Shafigh *et al.* (2016) revealed that a repeated call to the first step can be the main source of a computational hurdle. In order to shed light on this issue, let us first consider the implementation of the largest set of constraints (Eq. (18)) of the MILP model in solving this model using an exact method such as branch-and-bound (B&B) algorithm. This constraint has to be added into a modeling environment following a pseudopod in Figure 4 with no apparent alternative. In this figure, it can be seen that the implementation of this constraint involves $M^2 \times (R-1)(R) \times J^2 \times S \times O^2$ number of nested loops (assuming $R_m = R,\ S_j = S,\ O_j = O; \forall(m,j)$). For a small size problem with $M = 5$, $R = 15$, $J = 5$, $S = 3$, and $O = 4$, the number of loops involved is 6,300,000 which is quite large. And this process may take up to several minutes depending on the size of the problem, but it contributes less to the overall computational challenge as this process happens only once at the beginning of the B&B algorithm. However, as it was stated above, the process of adding an objective function and constraints is repeated for a very large number of trial solutions in the LP-A-GA. Hence, it can hinder the hybridization of the GA with the linear programming if it is not implemented creatively.

One possible implementation of the LP-subproblem is partially depicted in Figure 5, showing the implementation of only Eq. (37). We refer to this approach as a direct approach (Approach-1) since it follows the same "for-loop" structure as the implementation of the corresponding constraint in the MILP shown in Figure 4. In this approach all the integer variables are first decoded from the solution chromosome under consideration and their values are used to determine the conditions in which the constraints of the LP-subproblem are applicable. A simple comparison between Figures 4 and 5 clearly indicates that Approach-1 involves the same large number of loops as that of the MILP implementation. Thus, a repeated call to this approach is computationally prohibitive. To alleviate this computational difficulty, we develop an alternative approach (Approach-2) that drastically reduces the number of "for-loops" required to populate the constraints of the LP-subproblem. The pseudocode for this approach is partially depicted in Figure 6. In this approach, we first define two data structures: (i) {**Machine**$[m]$.**Run**$[r]$.*JobIndex* or *SubloIndex* or *OperationsIndex*} and (ii) {**Job**$[j]$.**Sublot**$[s]$.**Opertions**$[o]$.*MachineIdex* or *RunIndex*}. The former can be dot operated to store (or access) the *JobIndex*, *SubloIndex* or *OperationsIndex* of the process assigned to the r^{th} run of machine m, whereas the later can be dot operated to store(or access) the *MachineIdex* or *RunIndex* in which operation o of sublot s of job j is processed.

Before the for-loop is started, the integer variable $\gamma_{s,j}$ for each (s,j) is decoded from the chromosome and a run counter $RC[m]$ is set to zero for each machine m. Then, the for-loop scans the genes in RHS-segment of the chromosome for $g = 1$ to $TotalNummerOfOperations$ (see Figure 2 where g runs from 1 to $TotalNummerOfOperations = \sum_j S_j \times O_j = 22$). At a particular stage g of the loop, the indices (j,s,o,m) are obtained from gene(g) of the chromosome. The run counter for machine m is increased by one and the value is passed to the index r. For $\gamma_{s,j} > 1$, this corresponds to $x_{r,m,o,s,j} = 1$ and $y_{r,m,o,j} = 1$ and the corresponding indices (j,s,o,m,r) are stored in the appropriate data structure. Now, if $r > 1$ and $o > 1$, the indices (r',m',j',o') that correspond to $y_{r-1,m,o',j'} = 1$ and $x_{r',m',o-1,s,j} = 1$ are retrieved from the data that were recorded in precious stage of the for-loop. Once the indices that correspond to $y_{r-1,m,o',j'} + x_{r,m,o,s,j} + x_{r',m',o-1,s,j} = 3$ are identified, the constraint is added to the LP-subproblem. By following this approach, the total number of loops required to populate all the constraints of the LP-subproblem is reduced to $\sum_j S_j \times O_j$ (which

equal to $J \times S \times O$ assuming $S_j = S$ and $O_j = O$ for each job j). For the values of M, R, J, S, and O considered in the previous paragraph, the number of loops in Approach-2 is only $J \times S \times O = 60$ which is extremely low compared to 6,300,000 in Approach-1. This drastic reduction in the number of loops required to populate the constraints of the LP-subproblem enables the hybridization of the GA with linear programming. The complete pseudocodes for both Approach-1 and Approach-2 are given in the appendixes Appendix A and Appendix B, respectively.

for $\{m \leftarrow 1$ **to** $M\}$ **for** $\{r \leftarrow 2$ **to** $R_m\}$ **for** $\{m' \leftarrow 1$ **to** $M\}$ **for** $\{r' \leftarrow 1$ **to** $R_{m'}\}$ **for** $\{j \leftarrow 1$ **to** $J\}$ **for** $\{s \leftarrow 1$ **to** $S_j\}$ **for** $\{o \leftarrow 2$ **to** $O_j\}$ **for** $\{j' \leftarrow 1$ **to** $J\}$ **for** $\{o' \leftarrow 1$ **to** $O_{j'}\}$ **do**
  ADD-Constraint Eq. (18):
  $\widehat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m,o',j'} \cdot A_{o,j} - \Omega \cdot (y_{r-1,m,o',j'} + x_{r,m,o,s,j} + x_{r',m',o-1,s,j}) + 3\Omega \geq \widehat{c}_{r',m'} + L_{o,j}$
end

Figure 4: Pseudocode for populating constraint Eq. (18) into the MILP-model

Input: For the solution under consideration decode the values of the integer variables $x_{r,m,o,s,j}$, $y_{r,m,o,j}$, $\gamma_{s,j}$, $z_{r,m}$ for all (r, m, o, s, j)

---

for $\{m \leftarrow 1$ **to** $M\}$ **for** $\{r \leftarrow 1$ **to** $R_m\}$ **for** $\{m' \leftarrow 1$ **to** $M\}$ **for** $\{r' \leftarrow 1$ **to** $R_{m'}\}$ **for** $\{j \leftarrow 1$ **to** $J\}$ **for** $\{s \leftarrow 1$ **to** $S_j\}$ **for** $\{o \leftarrow 2$ **to** $O_j\}$ **for** $\{j' \leftarrow 1$ **to** $J\}$ **for** $\{o' \leftarrow 1$ **to** $O_{j'}\}$ **do**
  if $\{(y_{r-1,m,o',j'} + x_{r,m,o,s,j} + x_{r',m',o-1,s,j} = 3)$ **and** $\gamma_{s,j} = 1\}$ **then**
    ADD-Constraint Eq. (37): $\widehat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m,o',j'} \cdot A_{o,j} - \geq \widehat{c}_{r',m'} + L_{o,j}$
  end
end

Figure 5: Pseudocode for Approach-1 of implementing constraint Eq. (37) of the LP-Subproblem

3.5. Genetic operators

Genetic operators make the population evolve towards promising regions of the search space. Their design is so crucial as the convergence behaviour of the algorithm largely depends on them. Five crossover and six mutation operators were designed for the pure GA in Defersha and Chen (2012). As the solution representations of the pure GA and LP-A-GA are identical, all the operators of the pure GA are applicable in LP-A-GA as well. Most of the operators are also directly applicable in the LP-E-GA with few minor exceptions. The Sublot Step Mutation (SStM) operator, in pure GA and LP-A-GA, is applied on each gene $\alpha_{s,j}$ in the LHS-Segment to step up or down the value of this gene while keeping its value between 0 and 1. However, in LP-E-GA, the gene $\alpha_{s,j}$ can assume only a binary value (0 or 1). Hence SStM is modified in LP-E-GA in such a way that it only flips the value of $\alpha_{s,j}$ from 0 to 1 or vice versa with a small probability. The Sublot Swap Mutation (SSwM) is applied on each j in the LHS-Segment to swap the values of two arbitrarily selected genes $\alpha_{s,j}$ and $\alpha_{s',j}$. However, in LP-E-GA, there is a greater chance for a pair of $\alpha_{s,j}$ and $\alpha_{s',j}$ to be both equal to zero or one. Hence, the swap is performed whenever the two values are not identical.

In addition to crossover and mutation operators, the other important operator in applying a GA is the *selection operator*. It is needed to simulate the natural selection process in GAs in which individual solutions are chosen from a population to form the next generation following the principles of the survival of the fittest and probability. This operator depends neither on the solution representation nor on the evaluation function and as such the same selection operator can be applied across different GAs. The selection operator applied in the LP-A-GA and the LP-E-GA is the $k-$ways tournament selection which is similar to that used in the Pure GA in Defersha and Chen (2012). In this operator, k individuals are randomly selected and the one presenting the highest

Input: For the solution under consideration decode the values of the integer variable $\gamma_{s,j}$.
Input: For each machine m, set its run counter $RC[m] = 0$.

---

for $\{g \leftarrow 1$ **to** $TotalNumberOfOperations\}$ **do**
  $(j, s, o, m) \leftarrow gene(g)$
  if $\{\gamma_{s,j} = 1\}$ **then**
    $RC[m] \leftarrow RC[m] + 1$
    $r \leftarrow RC[m]$
    Machine$[m]$.**Run**$[r]$.$JobIndex \leftarrow j$
    Machine$[m]$.**Run**$[r]$.$SublotIndex \leftarrow s$
    Machine$[m]$.**Run**$[r]$.$OperationsIndex \leftarrow o$
    Job$[j]$.**Sublot**$[s]$.**Operation**$[o]$.$MachineIndex \leftarrow m$
    Job$[j]$.**Sublot**$[s]$.**Operation**$[o]$.$RunIndex \leftarrow r$

    ---

    if $\{r > 1$ **and** $o > 1\}$ **then**
      $m' =$ **Job**$[j]$.**Sublot**$[s]$.**Operation**$[o-1]$.$MachineIndex$
      $r' =$ **Job**$[j]$.**Sublot**$[s]$.**Operation**$[o-1]$.$RunIndex$
      $j' =$ **Machine**$[m]$.**Run**$[r-1]$.$JobIndex$
      $o' =$ **Machine**$[m]$.**Run**$[r-1]$.$OperationIndex$
      ADD-Constraint Eq. (37): $\widehat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m,o',j'} \cdot A_{o,j} - \geq \widehat{c}_{r',m'} + L_{o,j}$
    end
  end
end

Figure 6: Pseudocode for Approach-2 of implementing constraint Eq. (37) of the LP-Subproblem

fitness (smallest makespan) is declared the winner and a copy of this individual is added to the mating pool to form the next generation. Then, the k individuals in the tournament are placed back to the current population and the process is repeated. This continues until the number of individuals added to the mating pool is equal to the population size.

3.6. Steps of the GAs

The steps of a typical GA are: (1) initialize population, (2) evaluate fitness, (3) generate new population, (4) replace old population by new population, and repeat steps 2, 3 and 4 until an end condition is satisfied. The GAs considered in this paper (Pure GA, LP-E-GA, and LP-A-GA) follow these general steps with slight variations as indicated below. ***Initialization:*** The gene $\alpha_{s,j}$ in the LHS-Segment is initialized with a real number between 0 and 1 in the case of pure GA and LP-A-GA, whereas in the case of LP-E-GA this gene assumes only 0 or 1. In LP-A-GA, ϕ% percent of the initial population are randomly selected and improved by solving their corresponding LP-subproblem. ***Evaluation:*** This step is performed solely by following the procedure presented in Section 3.2 in pure GA, and by solving a LP-subproblem in LP-E-GA. Evaluation in LP-A-GA is primarily done similar to that in pure GA with the exception that in every H number of generations the top ϕ% percent of the population are further refined by solving LP-subproblems. The algorithms were coded in C++ where a linear programming solver in ILOG CPLEX (IBM, 2015) was interfaced to solve the LP-subproblems

Steps of the Pure GA

Step 1. Randomly generate initial population such that a gene $\alpha_{s,j}$ in the LHS-Segment assumes a ***real value*** between 0 and 1. Set a generation counter $G = 1$.

Step 2. Evaluate each individual solution using the procedure presented in Section 3.2 (Evaluation in pure GA). Update the best solution so far found.

Step 3. Apply genetic operators (Selection, Crossover and Mutation) to create a new population. Set $G = G + 1$.

Step 4. Replace the old population by the newly generated population for the next run of the algorithm

Step 5. If $G = G_{max}$, stop; otherwise go to Step-2.

Steps of LP-Embedded-GA

Step 1. Randomly generate initial population such that a gene $\alpha_{s,j}$ in the LHS-Segment assumes a ***binary value*** (0 or 1). Set a generation counter $G = 1$.

Step 2. Evaluate each individual solution by solving the corresponding LP-sublproblem. Update the best solution so far found.

Step 3. Apply genetic operators (Selection, Crossover and Mutation) to create the new population. Set $G = G + 1$.

Step 4. Replace the old population by the newly generated population for the next run of the algorithm

Step 5. If $G = G_{max}$, stop; otherwise go to Step-2.

Steps of LP-Assisted-GA

Step 1. Randomly generate initial population such that a gene $\alpha_{s,j}$ in the LHS-Segment assumes a ***real value*** between 0 and 1. Apply LP-improvement on $\phi\%$ of randomly selected individuals in the initial population. Set a generation counter $G = 1$.

Step 2. Evaluate each individual solution using the procedure presented in Section 3.2 (Evaluation in pure GA). If $G \mod H = 0$, go to Step-3; Otherwise, go to Step-4. Update the best solution so far found.

Step 3. Sort the individuals in the population in decreasing order of their fitness (increasing order of makespan). Apply LP-improvement on the top $\phi\%$ of the population.

Step 4. Apply genetic operators (Selection, Crossover and Mutation) to create the new population. Set $G = G + 1$.

Step 5. Replace the old population by the newly generated population for the next run of the algorithm

Step 6. If $G = G_{max}$, stop; otherwise go to Step-2.

4. Numerical illustrations

This section presents numerical examples to illustrate the solution enhancement that can be achieved using linear programming and compare its different implementation approaches and hybridization levels. Empirical studies on the computational performances of pure and hybrid GAs and their relative robustness to parameter settings are also presented. Readers interested in the numerical illustrations of the features of the proposed model such as lot streaming, machine release date, lag time and the nature of setup being attached/detached are referred to Defersha and Chen (2012).

4.1. Solution improvement using LP

In the pure GA, both the continuous and discrete variables are determined by the stochastic search alone. The search space is very large as there are infinite combinations of the values of the continuous variables corresponding to every possible set of values of the integer variables. This makes finding a good solution within a reasonable amount of time very difficult. In linear programming hybridized GA, the continuous variables which optimally correspond to a particular integer solution can be determined by solving a LP-subproblem. In order to illustrate the improvements that can be achieved using this approach on a particular solution and on a randomly generated population, we consider a scheduling problem (with lot streaming) in a small flexible job-shop consisting of three jobs and four machines. The batch size and maximum number of sublots of each job are given in Table 3 along with, for each operation, (i) the nature of the setup (attached or detached), (ii) lag time, (iii) alternative machines, and (iv) the corresponding processing times. The sequence-dependent setup time data are given in Table 4. The release dates of the machines were assumed to be $D_1 = D_2 = D_4 = 1080$ minutes and $D_3 = 0$.

For this small problem, an arbitrary solution was generated based on the solution representation in Figure 3. The solution provides the assignment and sequencing of the various operations on each machine. The sizes of the sublots were determined using Eq. (30) and the makespan of the resulting schedule was evaluated by applying the procedure described in Section 3.2. This same solution was further improved, with same operation assignment and sequencing (same integer solution), by solving the corresponding LP-subproblem to determine the optimal sublot sizes and makespan. The resulting schedules, before and after improvement, are depicted in Figure 7 (the numerical values of the starting and the ending times of the setups and operations are detailed in Table 5). As can be seen from these solutions, the values of makespan of the schedules before and after improvement are 22336 and 19823 minutes, respectively. This is about 11% reduction in the makespan on the arbitrarily generated solution achieved by solving a LP-subproblem. To illustrate the improvement that can be achieved on an entire initial papulation, we generate a population of 5000 individuals for the problem described above. The makespan corresponding to each individual solution was first evaluated by applying the procedure outlined in Section 3.2. Then each individual was further refined by solving the corresponding LP-subproblem. The distributions of the makespan of the population before and after the improvement are shown in Figure 8. From this figure, it can be seen that the average makespan of the population was reduced from 24732 to 19452 minutes which represents a 21% average improvement. Moreover, from the frequency of the distributions, it is possible to see that the number of good quality solution has been substantially increased.

Table 3: Processing Data for Jobs

					Alternative routes, $(m, T_{o,j,m})$			
j	B_j	S_j	o	$A_{o,j}$	$L_{o,j}$	1	2	3
1	1240	3	1	na	na	(1, 6.00)	(3, 5.25)	
			2	1	0	(1, 4.50)	(2, 5.00)	
			3	0	0	(1, 2.50)	(3, 2.75)	(4, 2.75)
2	1480	3	1	na	na	(3, 5.50)	(4, 5.75)	
			2	1	0	(1, 3.50)	(2, 3.25)	(4, 3.50)
3	1290	3	1	na	na	(1, 4.50)	(4, 4.75)	
			2	1	80	(1, 5.50)	(3, 5.75)	(4, 5.25)
			3	1	0	(1, 6.50)	(3, 6.50)	(4, 6.75)

na = not applicable

Table 4: Sequence Dependent Setup Time Data

j	o	m	Setup time $(S^*_{o,j,m})$, $(j', o', S_{o,j,m,o',j'})$ $\cdots$
1	1	1	(150), (1,1,80), (1,2,160), (1,3,160), (2,2,270), (3,1,270), (3,2,240), (3,3,210)
		3	(200), (1,1,80), (1,3,160), (2,1,210), (3,2,240), (3,3,210)
	2	1	(50), (1,1,120), (1,2,60), (1,3,140), (2,2,150), (3,1,180), (3,2,240), (3,3,300)
		2	(100), (1,2,80), (2,2,180)
	3	1	(150), (1,1,120), (1,2,160), (1,3,60), (2,2,270), (3,1,270), (3,2,270), (3,3,210)
		3	(150), (1,1,140), (1,3,60), (2,1,180), (3,2,210), (3,3,270)
		4	(100), (1,3,60), (2,1,240), (2,2,180), (3,1,270), (3,2,270), (3,3,270)
2	1	3	(200), (1,1,180), (1,3,300), (2,1,70), (3,2,240), (3,3,270)
		4	(100), (1,3,210), (2,1,70), (2,2,160), (3,1,180), (3,2,180), (3,3,180)
	2	1	(100), (1,1,180), (1,2,210), (1,3,210), (2,2,70), (3,1,150), (3,2,300), (3,3,180)
		2	(150), (1,2,180), (2,2,80)
		4	(150), (1,3,180), (2,1,100), (2,2,60), (3,1,270), (3,2,270), (3,3,270)
3	1	1	(100), (1,1,210), (1,2,210), (1,3,210), (2,2,240), (3,1,80), (3,2,180), (3,3,180)
		4	(100), (1,3,210), (2,1,240), (2,2,240), (3,1,60), (3,2,200), (3,3,120)
	2	1	(100), (1,1,180), (1,2,300), (1,3,210), (2,2,270), (3,1,140), (3,2,50), (3,3,140)
		3	(200), (1,1,270), (1,3,270), (2,1,300), (3,2,60), (3,3,180)
		4	(150), (1,3,180), (2,1,270), (2,2,240), (3,1,180), (3,2,80), (3,3,120)
	3	1	(100), (1,1,270), (1,2,180), (1,3,150), (2,2,180), (3,1,160), (3,2,160), (3,3,70)
		3	(100), (1,1,180), (1,3,180), (2,1,270), (3,2,180), (3,3,80)
		4	(50), (1,3,270), (2,1,180), (2,2,150), (3,1,180), (3,2,140), (3,3,70)

Table 5: The details of the schedules shown in Figure 7

		Before Solution Improvement				After Solution Improvement			
Machine	Run	(j, s, o)	SB	SE/PB	PE	(j, s, o)	SB	SE/PB	PE
M1	R1	(1,2,2)	3235.6	3285.6	5887.6	(1,2,2)	1080	1130	1884.3
	R2	(3,1,1)	5887.6	6097.6	8628.6	(3,1,1)	1884.3	2094.3	4620.7
	R3	(3,3,2)	8628.6	8768.6	12016.5	(3,3,2)	17028.5	17168.5	17882.6
M2	R1	(1,3,2)	6790	6890	10198.9	(1,3,2)	6790.0	6890.0	12251.9
	R2	(2,2,2)	13658.2	13838.2	17790.4	(2,2,2)	14753.6	14933.6	19533
	R3	(2,3,2)	21398.4	21478.4	22336.3	(2,3,2)	19533	19613	19823.6
M3	R1	(1,2,1)	0.0	200	3235.6	(1,2,1)	0.0	200	1080
	R2	(1,3,1)	3235.6	3315.6	6790.0	(1,3,1)	1080	1160	6790
	R3	(2,2,1)	6790	6970	13658.2	(2,2,1)	6790	6970	14753.6
	R4	(1,3,3)	13658.2	13838.2	15658.1	(1,3,3)	14753.6	14933.6	17882.6
	R5	(3,3,3)	15658.1	15838.1	19676.6	(3,3,3)	17882.6	18062.6	18906.6
	R6	(2,3,1)	19676.6	19946.6	21398.4	(2,3,1)	18906.6	19176.6	19533
M4	R1	(3,3,1)	1080.0	1180.0	3985.1	(3,3,1)	1080	1180	1796.7
	R2	(3,2,1)	3985.1	4045.1	4696	(3,2,1)	1796.7	1856.7	4700.7
	R3	(3,1,2)	8708.6	8888.6	11841.3	(3,1,2)	4700.7	4880.7	7828.2
	R4	(3,2,2)	11841.3	11921.3	12640.7	(3,2,2)	7828.2	7908.2	11051.6
	R5	(3,1,3)	12640.7	12780.7	16577.1	(3,1,3)	11051.6	11191.6	14981.2
	R6	(3,2,3)	16577.1	16647.1	17572.1	(3,2,3)	14981.2	15051.2	19092.6
	R7	(1,2,3)	17572.1	17842.1	19432.2	(1,2,3)	19092.6	19362.6	19823.6

Note: SB, SE, PB, PE stand for setup begins, setup ends, processing begins, and processing ends, respectively.

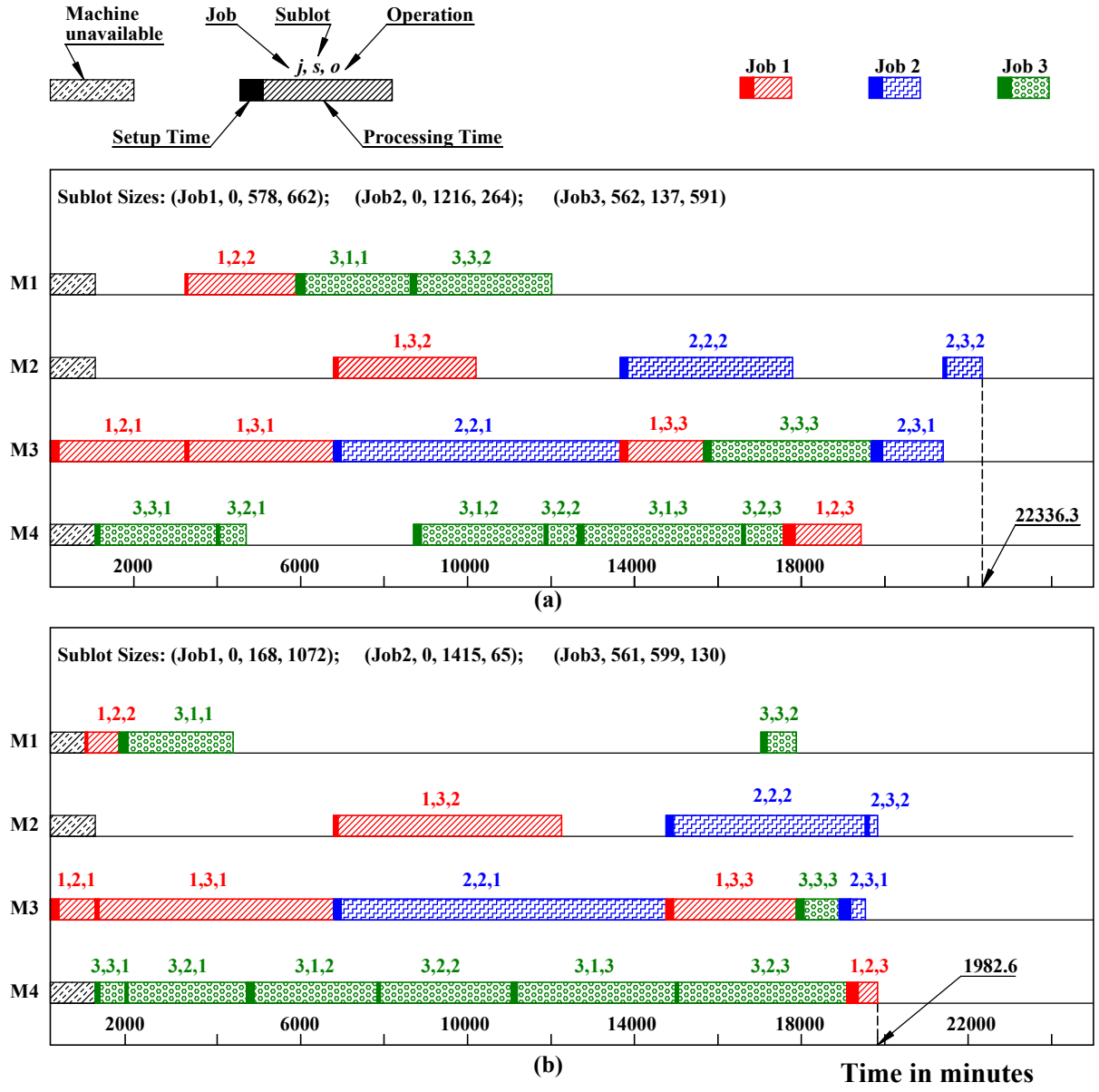


Figure 7: An arbitrary generated solution for problem-1: (a) before and (b) after improvement by solving a LP-subproblem. The detail numerical values are given in Table 5.

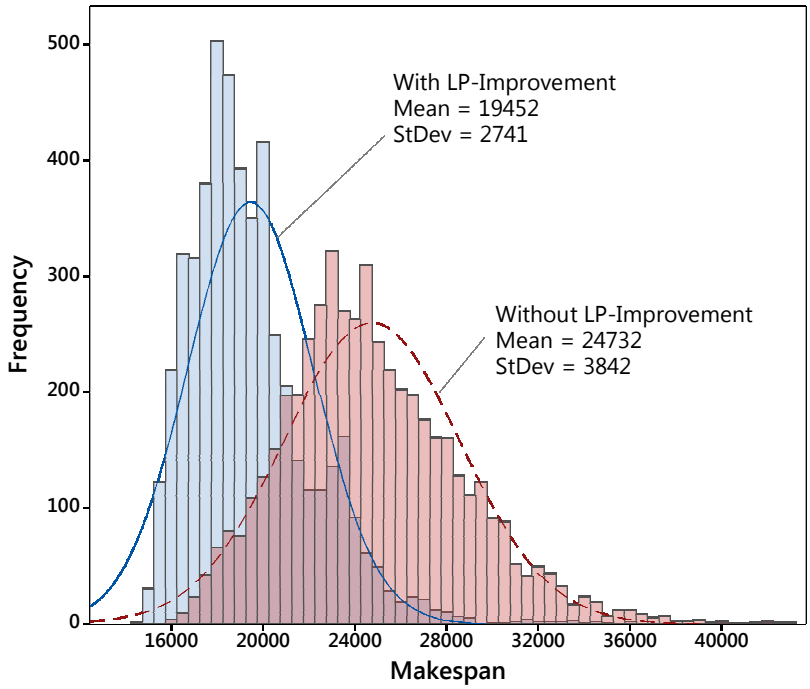


Figure 8: Distribution of makespan for an initial population of 5000 with and without LP-Improvement

4.2. LP-implementation approach-1 vs approach-2

In Section 3.4, we presented two approaches for populating the objective function and the constraints of the LP-subproblem. In this section, we exemplify the computational challenge in using approach-1 and the improvement obtained by using approach-2. In order to illustrate this improvement, problems of larger dimensions than Problem 1 (presented in the previous section) are considered. The general nature of the considered problems is depicted in Table 6. Table 7 shows the average modeling and solver times (in milliseconds) required in a single LP-improvement in 50 arbitrarily generated solutions in each of the four problems. These computational times were evaluated for both approaches. From this table, it can be seen that the modeling times are many times larger than the solver times in approach-1. Approach-2 eliminates this computational difficulty to almost nonexistent level. Without this improvement, the remarkable performance of the GA achieved (both in terms of solution quality and computational time) through hybridization with linear programming would be impossible. Thus, all the computational analysis in the remaining subsections are using approach-2.

4.3. LP-E-GA vs LP-A-GA

LP-E-GA requires every solution in every generation to be evaluated by solving a LP-subproblem. LP-A-GA, on the other hand, uses linear programming only to refine promising solutions periodically. Figure 9 shows the convergence behaviours in the first 250 generations of LP-E-GA, LP-A-GA and pure GA in solving problems 2 and 3 with a population size of 5000. In LP-A-GA(10) and LP-A-GA(50), 1% of the top best individuals in the population are improved using linear programming in every 10 and 50 generations, respectively. The convergence graphs show that both LP-E-GA and LP-A-GA provide comparable solution quality improvements over those that can be obtained by pure GA in both problems. However, the LP-A-GA require much shorter time compared to LP-E-GA. In running the 250 generations, LP-A-GA requires only 1 minute in problem 2 and 2.5

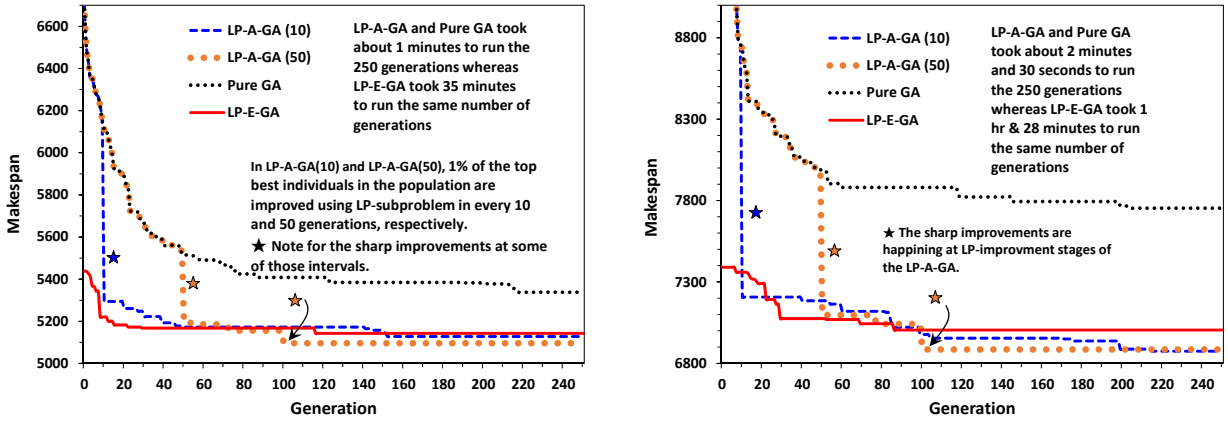
Table 6: The general nature of the problems considered

Problem No.	Number of machines	Number of jobs	Number of sublots for each job	Number of operations for the jobs	Number of alternative routes for the operation
2	8	20	4	3 to 5	1 to 3
3	12	30	4	3 to 6	1 to 3
4	10	25	4	3 to 4	1 to 3
5	12	35	3	2 to 4	1 to 3

Table 7: Average modeling and solver time in milliseconds for a single LP-evaluation in Approaches 1 and 2

	Approach 1		Approach 2	
Problem	Modeling time	Solver time	Modeling time	Solver time
2	3839.24	10.944	0.965	11.775
3	67953.60	37.898	1.875	41.258
4	8012.99	14.801	1.238	15.540
5	8361.72	9.2110	1.182	10.027

minutes in problem 3, whereas the LP-E-GA needs 35 and 88 minutes (about 3500% that of LP-A-GA computational times) to run the same number of generations in the two problems respectively. In addition to this tremendous speedup, the LP-A-GA also provides slightly better solutions than the LP-E-GA. This may be because the LP-A-GA has better local improvement ability by refining only promising solutions which may lead these refined solutions to dominate the population and the search to be intensified around these solutions.



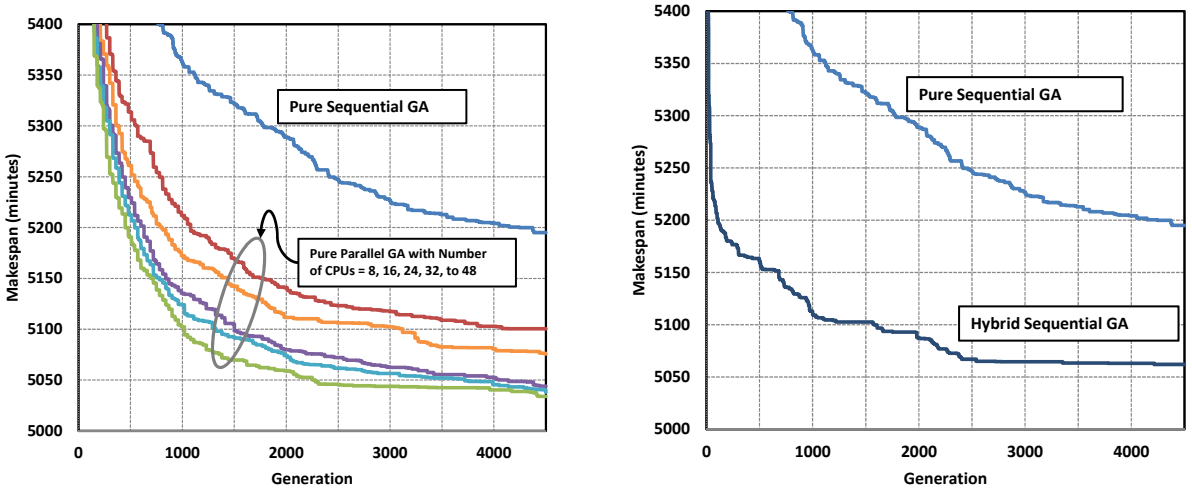
(a) (b)

Figure 9: Comparison of pure GA, LP-E-GA and LP-A-GA in solving (a) Problem 2 and (b) Problem 3

4.4. Parallelization vs Hybridization

This section compares the performance improvements of the GA achieved through parallelization and hybridization. In particular, the convergence rates and final solution qualities from pure sequential GA and pure parallel GA reposted in Defersha and Chen (2012) are compared against those found using the proposed hybrid sequential GA (namely LP-A-GA). Figure 10(a) shows the improvement

in convergence rate and final solution quality that can be achieved through parallelization of the pure GA as the number of processors (CPUs) increases in solving problem 2. The makespan of the final solution, 5227 minutes, achieved by the sequential GA was reduced by 109, 125, 165, 170, and 183 minutes as the number of processors (CPUs) increases to 8, 16, 24, 32, and 48, respectively. On the other hand, as shown in Figure 10(b), the sequential LP-A-GA demonstrate a faster convergence rate and provides a final solution which is very comparable to that achieved by using the resource intensive parallel GA. More interestingly, the LP-A-GA provides even better solutions with quicker convergence in problems 3, 4 and 5 than the parallel GA. These results are depicted in Figure 11.



(a) Parallelization (Defersha and Chen, 2012)

(b) Hybridization (this paper)

Figure 10: Performance improvement in solving Problem 2: (a) by parallelization as the number of processors is increased from 8, 16, 24, 32, to 48; (b) by hybridization of the GA with linear programming

4.5. Robustness of algorithms

In this section, the impact of the genetic parameters on the quality of the final solution from SGA and the proposed HGA are assessed empirically. Figure 12 shows the plots of the makespan of the schedules obtained after 3000 iterations by SGA and the proposed hybrid GA under different test runs. The test runs are differentiated by the settings of their genetic parameters as it was described in Defersha and Chen (2012). Figure 12 shows the makespan of the final solutions and their averages from the ten test runs when problem 2 is solved using SGA and HGA. As it can be seen from this figure, HGA demonstrates more robustness than the pure SGA against the changes in the genetic parameters. In HGA, 8 from 10 final solutions lie within plus or minus 1% of mean of the final solutions. However, in the SGA case, only 5 out of 10 answers lie within the same range of the average of its final solutions. This makes parameter tuning, a very essential task in using GAs, much easier in HGA than in pure GA.

5. Discussion and conclusion

A very large number of documents reporting the applications of metaheuristics have been published from a wider range of disciplines. However, pure applications of metaheuristics can be limiting when solving complex problems. Hence, many researchers developed hybrid metaheuristics where the metaheuristics are hybridized with other metaheuristics or OR/AI techniques. The hybridization of metaheuristics with linear programming (LP) is one of the many hybridization techniques

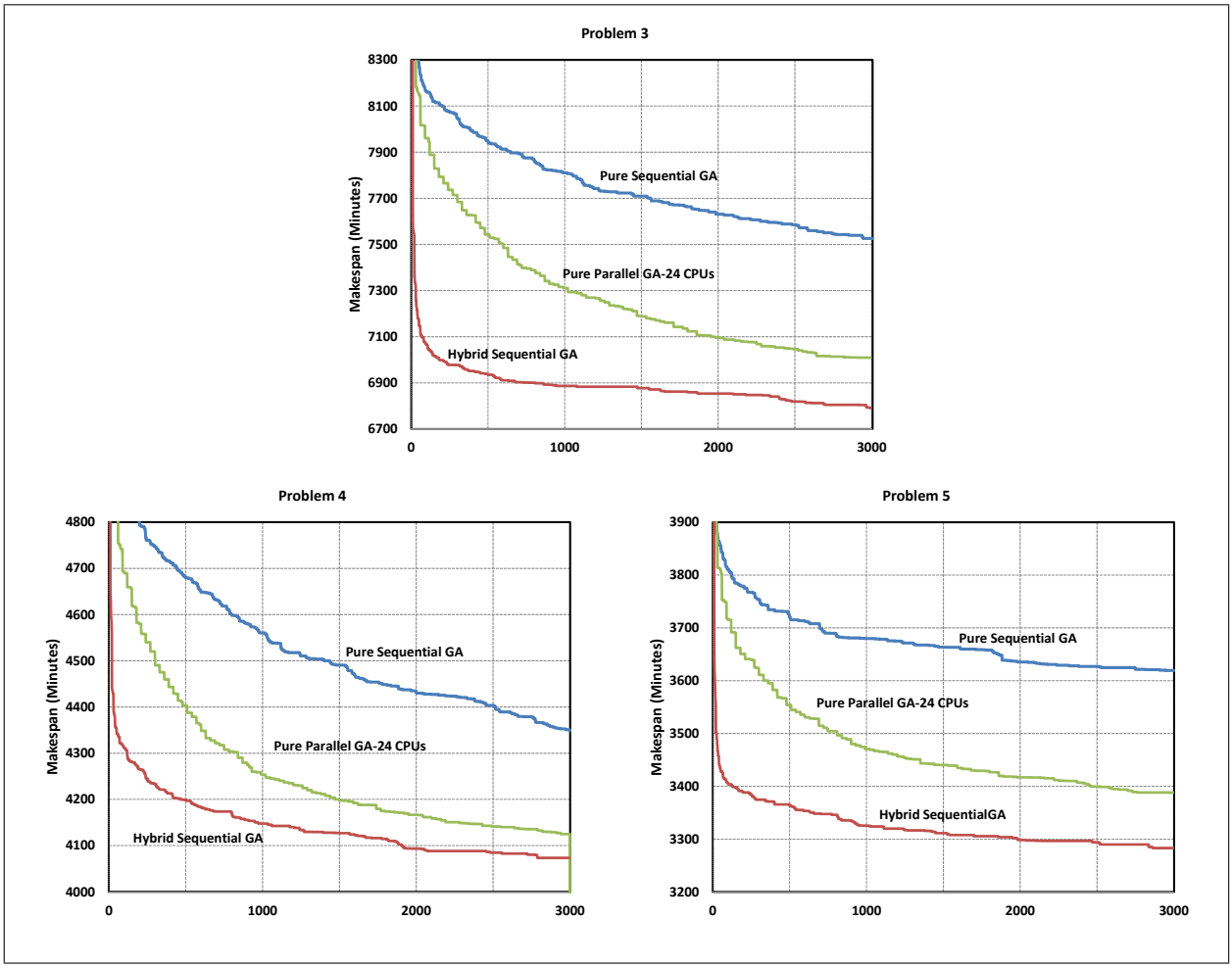


Figure 11: Average convergence of the SGA, PGA with 24 processor and HGA for problems 3, 4, and 5.

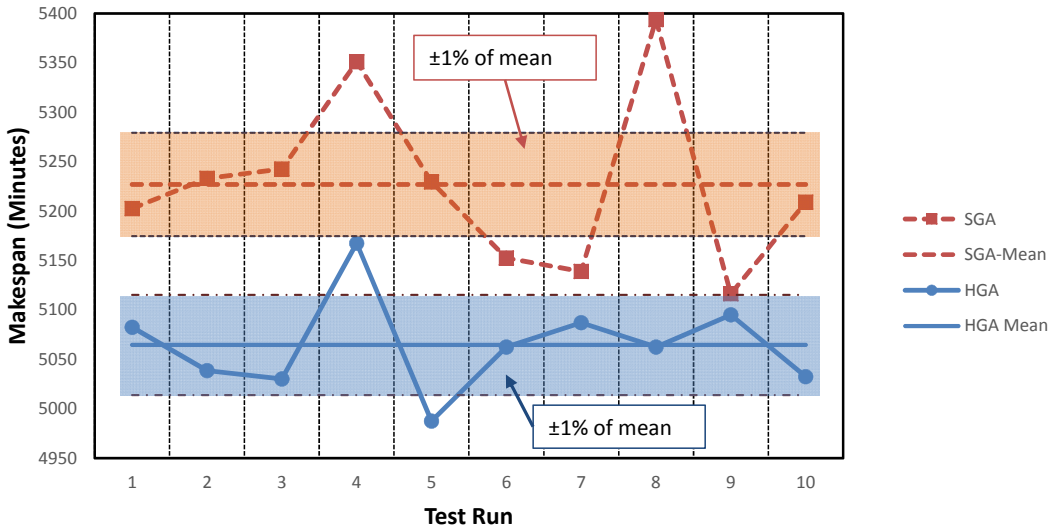


Figure 12: The effect of changing genetic parameters on the final solution quality obtained by the SGA and HGA in solving problem 2

reported. In this paper, we developed a LP hybridized metaheuristic based on genetic algorithm to solve a comprehensive MILP model in flexible jobshop scheduling problem with lot stream. Further, we distinguished LP-E-MH and LP-A-MH as two levels of hybridizations in solving MILP models when a LP-subproblem has to be repeatedly solved during the search process. In LP-E-MH based algorithms, the metaheuristic searches over the integer variables and calls a LP solver for every individual solution visited to find the corresponding values of the continuous variables. However, this can be computationally difficult when solving large size problems. To alleviate this problem, we proposed a LP-A-MH as an alternative where the metaheuristic has the capability to search both the integers and the continuous variables without the LP solver. Instead, the LP solver is used to assist the metaheuristic by further refining only promising solutions after every certain number of iterations through determining the optimal values of the continuous variables corresponding to those promising solutions. The proposed LP-A-GA approach drastically reduce computational time and also provides a comparable or better solution than LP-E-GA. Moreover, the sequential LP-A-GA, utilizing a single computational resource, outperforms or equally performs as the resource intensive parallel pure genetic algorithm that uses multiple concurrently available computational resource. We also illustrate the potential challenge that one may encounter when developing a hybrid metaheuristic where a LP-subproblem has to be solved repeatedly. Solving a LP-subproblem involves populating a large number of constraints into a model (task-1) and solving the model using a LP solver (task-2). At first it may appear that task-2 can be the major source of a computational difficulty. However, if it is not implemented systematically, task-1 can be the major source of computational difficulty. Two approaches, a direct approach (approach-1) and an alternative approach (approach-2) for accomplishing task-1 were presented. The alternative approach drastically decreases the computational time, making the hybridization of LP with the GA a working algorithm in solving the FJSP lot streaming problem considered in this paper.

6. Future research

As it is noted in literature (see Raidl (2006)), metaheuristic algorithms can be hybridized with several other OR/AI techniques. One of such OR techniques is Benders' Decomposition. To illustrate this hybridization process consider the general MILP problem described in Eqs. 1-6. Benders' decomposition projects this MILP problem onto y-space by defining the optimal value function

$$\begin{aligned} v(y) &= q^T y + \min_x p^T x \\ \text{Subject to:} & \\ Ax &\leq e - By \\ x &\in \mathbb{R}^{n_1} \\ x &\geq 0 \end{aligned}$$

and restating the MILP problem as

$$\begin{aligned} &\min_y v(y) \\ \text{Subject to:} & \\ y &\in \mathbb{Z}^{n_2} \\ y &\geq 0 \end{aligned}$$

Using the LP dual theory, the minimization w.r.t. x in the definition of $v(y)$ can be replaced by a maximization over the dual space. Then by enumerating the finitely many dual extreme points,

$v(y)$ can be written as

$$v(y) = \max_{k=1,\ldots K}\{\alpha^k y + \beta^k\}$$

where α^k and β^k are determined by the k^{th} dual extreme point. If the LP has been solved only $K' < K$ times, we can define a function

$$v'(y) = \max_{k=1,\ldots K'}\{\alpha^k y + \beta^k\} \leq v(y)$$

which underestimates $v(y)$. Thus, the dual solutions can be used to direct the genetic search for optimal discrete variables. Reader are referred to Ming-Che Lai *et al.* (2012) and Ming-Che Lai *et al.* (2010) for detail examples of such hybridization with the context of vehicle routing and capacitated plant location problems, respectively. Our future research is to consider this alternative approach in solving the flexible jobshop scheduling problem presented in this paper. We also encourage researchers to consider this approached. Our future research also includes expanding hybrid genetic algorithm to encompass a multi-objective scheduling with lot steaming in flexible jobshop and other manufacturing settings.

References

Bayram, H. and Şahin, R., 2016. A comprehensive mathematical model for dynamic cellular manufacturing system design and linear programming embedded hybrid solution techniques. *Computers & Industrial Engineering*, **91**, 10–29.

Blackburn, J., 1991. Time-Based Competition. Business One Irwin, Burr Ridge, IL,

Blesa, M. J., Blum, C., Cangelosi, A., Cutello, V., DI NUOVO, A., Pavone, M., and Talbi, E. G. (Eds.), 2016. The 10th International Workshop on Hybrid Metaheuristics HM2016, June 8-10, 2016. Plymouth, UK.

Blum, C., Blesa Aguilera, M. J., Roli, A., and Samples, M. (Eds.), 2008. Hybrid Metaheuristics-An Emerging Approach to Optimization. Vol. 114. Springer-Verlag, Berlin, Germany.

Blum, C., Puchinger, J., Raidl, G. R., and Roli, A., 2011. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, **11 (6)**, 4135–4151.

Bockerstette, J. and Shell, R., 1993. Time Based Manufacturing. McGraw-Hill, New York,

Cao, D., Defersha, F. M., and Chen, M., 2009. Grouping operations in cellular manufacturing considering alternative routings and the impact of run length on product quality. *International Journal of Production Research*, **47 (4)**, 989–1013.

Chan, F., Wong, T., and Chan, P., 2008a. The application of genetic algorithms to lot streaming in a job-shop scheduling problem. *International Journal of Production Research*, **In press (DOI: 10.1080/00207540701577369)**.

Chan, F., Wong, T., and Chan, P., 2008b. Lot streaming for product assembly in job shop environment. *Robotics and Computer-Integrated Manufacturing*, **24**, 321–331.

Chan, F. T., Wong, T., and Chan, P., 2004. Equal size lot streaming to job-shop scheduling problem using genetic algorithms. In: Intelligent Control, 2004. Proceedings of the 2004 IEEE International Symposium on. IEEE, pp. 472–476.

Chang, J. H. and Chiu, H. N., 2005. A comprehensive review of lot streaming. *International Journal of Production Research,*, **43**, 1515–1536.

Chen, H., Ihlow, J., and Lehmann, C., 1999. A genetic algorithm for flexible job-shop scheduling. In: Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on. Vol. 2. IEEE, pp. 1120–1125.

Cisty, M., 2010. Hybrid genetic algorithm and linear programming method for least-cost design of water distribution systems. *Water Resour Manage*, **24**, 1–24.

Cotta, C., Talbi, E.-G., and Alba, E., 2005. 15 parallel hybrid metaheuristics. *Parallel Metaheuristics: A New Class of Algorithms*, **47**, 347.

CPAIOR Conferece series, n.d. International conference on integration of AI and OR techniques in constraint programming for combinatorial optimization problems. http://www.andrew.cmu.edu/user/vanhoeve/cpaior/.

Dauzere-Peres, S. and Lasserre, J., 1997. Lot streaming in job-shop scheduling. *Operations Research*, **45**, 584–595.

Defersha, F. M. and Chen, M., 2008. A linear programming embedded genetic algorithm for an integrated cell formation and lot sizing considering product quality. *European Journal of Operational Research*, **187**, 46–69.

Defersha, F. M. and Chen, M., 2009a. A coarse-grain parallel genetic algorithm for flexible job-shop scheduling with lot streaming. 12th IEEE International Conference on Computational Science and Engineering. Vancover, CADADA, August 29-31, pp. 201–208.

Defersha, F. M. and Chen, M., 2009b. A simulated annealing algorithm for dynamic system reconfiguration and production planning in cellular manufacturing. *International Journal of Manufacturing Technology and Management*, **17 (1-2)**, 103–124.

Defersha, F. M. and Chen, M., 2010a. A hybrid genetic algorithm for flowshop lot streaming with setups and variable sublots. *International Journal of Production Research*, **48 (6)**, 1705–1726.

Defersha, F. M. and Chen, M., 2010b. A parallel genetic algorithm for a flexible job-shop scheduling with a sequence dependent setups. *International Journal of Advanced Manufacturing Technology*, **49**, 263–279.

Defersha, F. M. and Chen, M., 2012. Jobshop lot streaming with routing flexibility, sequence-dependent setups, machine release dates and lag time. *International Journal of Production Research*, **50 (8)**, 2331–2352.

IBM, 2015. IBM ILOG CPLEX Optimization Studio CPLEX User's Manual, Version 12.6. Online Documentation.

Kacem, I., 2003. Genetic algorithm for the flexible jobshop scheduling problem. In the Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics. Washington, DC, pp. 3464–6469.

Kulturel-Konak, S., 2012. A linear programming embedded probabilistic tabu search for the unequal-area facility layout problem with flexible bays. *European Journal of Operational Research*, **223**, 614–625.

Luo, X., Yang, W., Kwong, C., Tang, J., and Tang, J., 2014. Linear programming embedded genetic algorithm for product family design optimization with maximizing imprecise part-worth utility function. *Concurrent Engineering: Research and Applications*, **22 (4)**, 309–319.

Maniezzo, V., Stützle, T., and Vob, S. (Eds.), 2009. Matheuristics: Hybridizing Metaheuristics and Mathematical Programming. Springer-Verlag, Berlin, Germany.

Matheuristics2006, 2006. First Workshop on Mathematical Contributions to Metaheuristics. http://astarte.csr.unibo.it/matheuristics2006/.

Ming-Che Lai, Han-suk Sohn, Tzu-Liang(bill) Tseng, and Bricker, Dennis L., 2012. A hybrid benders/genetic algorithm for vehicle routing and scheduling. *International Journal of Industrial Engineering*, **12 (1)**, 33–46.

Ming-Che Lai, Han-suk Sohn, Tzu-Liang(bill) Tseng, and Chunkun Chiang, 2010. A hybrid algorithm for capacitated plant location problem. *Expert Systems with Applications*, **37 (12)**, 8599–8605.

Neto, T. and Pedroso, J. P., 2003. Grasp for linear integer programming. In: In J. P. Sousa and M. G. C. Resende, editors, Metaheuristics: Computer Decision Making, Combinatorial Optimization Book Series. Kluwer Academic Publishers, p. 545574.

Pedroso, J. P., 2005. Tabu search for mixed integer programming. In: In C. Rego and B. Alidaee, editors, Metaheuristic Optimization via Memory and Evolution. Springer-Verlag, Berlin, Germany, pp. 247–261.

Potts, C. and Baker, K., 1989. Flow shop scheduling with lot streaming. *Operations Research Letter*, **8**, 297–303.

Puchinger, J. and Raidl, G. R., 2005. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In: Artificial intelligence and knowledge engineering applications: a bioinspired approach. Springer, pp. 41–53.

Raidl, G. R., 2006. A unified view on hybrid metaheuristics. In: Hybrid Metaheuristics. Springer, pp. 1–12.

Raidl, G. R. and Puchinger, J., 2008. Cobining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In: Blum, C. and Blesa Aguilera, M. J., and Roli, A. and Samples, M. (Eds): Hybrid Metaheuristics-An Emerging Approach to Optimization. Springer-Verlag, Berlin, Germany, pp. 31–62.

Reis, L. F. R., Walters, G. A., Savic, D., and Chaudhry, F. H., 2005. Multi-reservoir operation planning using hybrid genetic algorithm and linear programming (GA-LP): An alternative stochastic approach. *Water Resources Managment*, **19**, 831–848.

Reiter, S., 1966. A system for managing job shop production. *Journal of Business*, **34**, 371–393.

Rezazadeh, H., Mahini, R., and Zarei, M., 2011. Solving a dynamic virtual cell formation problem by linear programming embedded particle swarm optimization algorithm. *Applied Soft Computing*, **11**, 3160–3169.

Shafigh, F., Defersha, F. M., and Moussa, S. E., 2016. A linear programming embedded simulated annealing in the design of distributed layout with production planning and systems reconfiguration. *International Journal of Advanced Manufacturing Techonolgy*, **DOI 10.1007/s00170-016-8813-z**.

Talbi, E.-G., 2002. A taxonomy of hybrid metaheuristics. *Journal of heuristics*, **8 (5)**, 541–564.

Talbi, E. G. (Ed.), 2013. Hybrid Metaheuristics. Studies in Computational Intelligence. Springer-Verlag, Berlin, Germany.

Teghem, J., Pirlot, M., and Antoniadis, C., 1995. Embedding of linear programming in a simulated annealing algorithm for solving a mixed integer production planning problem. *Journal of Computational and Applied Mathematics*, **64**, 91–102.

Urdaneta, A. J., Gómez, J. F., Sorrentino, E., Flores, L., and Diaz, R., 1999. A hybrid genetic algorithm for optimal reactive power planning based upon successive linear programming. *IEEE Transactions on Power Systems*, **14 (4)**.

Zhang, H. and Gen, M., 2005. Multistage-based genetic algorithm for flexible job-shop scheduling problem. *Journal of Complexity International*, **11**, 223–232.

APPENDIX

Appendix A. Complete pseudocode for Approach-1

Input: For the solution under consideration decode the values of the integer variables $x_{r,m,o,s,j}$, $y_{r,m,o,j}$, $\gamma_{s,j}$, $z_{r,m}$ for all (r, m, o, s, j)

ADD-Objective Minimize Eq. (31): $Objective = c_{max}$;

---

for $\{m \leftarrow 1$ **to** $M\}$ **for** $\{j \leftarrow 1$ **to** $J\}$ **for** $\{s \leftarrow 1$ **to** $S_j\}$ **for** $\{o \leftarrow 1$ **to** $O_j\}$ **do**
  if $\{\gamma_{s,j} = 1\}$ **then**
    ADD-Constraint Eq. (32): $c_{max} \geq c_{o,s,j,m}$
  end
end

---

for $\{m \leftarrow 1$ **to** $M\}$ **for** $\{r \leftarrow 1$ **to** $R_m\}$ **for** $\{j \leftarrow 1$ **to** $J\}$ **for** $\{s \leftarrow 1$ **to** $S_j\}$ **for** $\{o \leftarrow 1$ **to** $O_j\}$ **do**
  if $\{x_{r,m,o,s,j} = 1\ AND\ \gamma_{s,j} = 1\}$ **then**
    ADD-Constraint Eq. (33): $\widehat{c}_{r,m} = c_{o,s,j,m}$
  end
end

---

for $\{m \leftarrow 1$ **to** $M\}$ **for** $\{j \leftarrow 1$ **to** $J\}$ **for** $\{s \leftarrow 1$ **to** $S_j\}$ **for** $\{o \leftarrow 1$ **to** $O_j\}$ **do**
  if $\{x_{1,m,o,s,j} = 1$ **and** $\gamma_{s,j} = 1\}$ **then**
    ADD-Constraint Eq. (34): $\widehat{c}_{1,m} - b_{s,j} \cdot T_{o,j,m} - S^*_{o,j,m} \geq D_m$
  end
end

---

for $\{m \leftarrow 1$ **to** $M\}$ **for** $\{r \leftarrow 2$ **to** $R_m\}$ **for** $\{j \leftarrow 1$ **to** $J\}$ **for** $\{s \leftarrow 1$ **to** $S_j\}$ **for** $\{o \leftarrow 1$ **to** $O_j\}$ **for** $\{j' \leftarrow 1$ **to** $J\}$ **for** $\{o' \leftarrow 1$ **to** $O_{j'}\}$ **do**
  if $\{y_{r-1,m,o',j'} + x_{r,m,o,s,j} = 2$ **and** $\gamma_{s,j} = 1\}$ **then**
    ADD-Constraint Eq. (35): $\widehat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m,o',j'} \geq \widehat{c}_{r-1,m}$
  end
end

---

for $\{m \leftarrow 1$ **to** $M\}$ **for** $\{m' \leftarrow 1$ **to** $M\}$ **for** $\{r' \leftarrow 1$ **to** $R_{m'}\}$ **for** $\{j \leftarrow 1$ **to** $J\}$ **for** $\{s \leftarrow 1$ **to** $S_j\}$ **for** $\{o \leftarrow 2$ **to** $O_j\}$ **do**
  if $\{(1, m) \neq (r', m')$ **and** $x_{1,m,o,s,j} + x_{r',m',o-1,s,j} = 2$ **and** $\gamma_{s,j} = 1\}$ **then**
    ADD-Constraint Eq. (36): $\widehat{c}_{1,m} - b_{s,j} \cdot T_{o,j,m} - S^*_{o,j,m} \cdot A_{o,j} - \geq \widehat{c}_{r',m'} + L_{o,j}$
  end
end

---

for $\{m \leftarrow 1$ **to** $M\}$ **for** $\{r \leftarrow 2$ **to** $R_m\}$ **for** $\{m' \leftarrow 1$ **to** $M\}$ **for** $\{r' \leftarrow 1$ **to** $R_{m'}\}$ **for** $\{j \leftarrow 1$ **to** $J\}$ **for** $\{s \leftarrow 1$ **to** $S_j\}$ **for** $\{o \leftarrow 2$ **to** $O_j\}$ **for** $\{j' \leftarrow 1$ **to** $J\}$ **for** $\{o' \leftarrow 1$ **to** $O_{j'}\}$ **do**
  if $\{y_{r-1,m,o',j'} + x_{r,m,o,s,j} + x_{r',m',o-1,s,j} = 3$ **and** $\gamma_{s,j} = 1\}$ **then**
    ADD-Constraint Eq. (37): $\widehat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m,o',j'} \cdot A_{o,j} - \geq \widehat{c}_{r',m'} + L_{o,j}$
  end
end

---

for $\{j \leftarrow 1$ **to** $J\}$ **for** $\{s \leftarrow 1$ **to** $S_j\}$ **do**
  ADD-Constraint Eq. (38): $b_{s,j} \leq B_j \times \gamma_{s,j}$
end

---

for $\{j \leftarrow 1$ **to** $J\}$ **do**
  ADD-Constraint Eq. (39): $\sum_{\forall s|(\gamma_{s,j}=1)} b_{s,j} = B_j$
end

Appendix B. Complete pseudocode for Approach-2

Input: For the solution under consideration decode the values of the integer variable $\gamma_{s,j}$.
Input: For each machine m, set its run counter $RC[m] = 0$.

ADD-Objective Minimize Eq. (31): $Objective = c_{max}$;

---

for $\{g \leftarrow 1$ **to** $TotalNumberOfProcesses\}$ **do**
  $(j, s, o, m) \leftarrow gene(g)$
  if $\{\gamma_{s,j} = 1\}$ **then**
    $RC[m] \leftarrow RC[m] + 1$
    $r \leftarrow RC[m]$
    Machine$[m]$.**Run**$[r].JobIndex \leftarrow j$
    Machine$[m]$.**Run**$[r].SublotIndex \leftarrow s$
    Machine$[m]$.**Run**$[r].OperationsIndex \leftarrow o$
    Job$[j]$.**Sublot**$[s]$.**Operation**$[o].MachineIndex \leftarrow m$
    Job$[j]$.**Sublot**$[s]$.**Operation**$[o].RunIndex \leftarrow r$

    ADD-Constraint Eq. (32): $c_{max} \geq c_{o,s,j,m}$

    ADD-Constraint Eq. (33): $\widehat{c}_{r,m} = c_{o,s,j,m}$

    ---

    if $\{r = 1\}$ **then**
      ADD-Constraint Eq. (34): $\widehat{c}_{1,m} - b_{s,j} \cdot T_{o,j,m} - S^*_{o,j,m} \geq D_m$
    end

    ---

    if $\{r > 1\}$ **then**
      $j' =$ **Machine**$[m]$.**Run**$[r-1].JobIndex$
      $o' =$ **Machine**$[m]$.**Run**$[r-1].OperationIndex$
      ADD-Constraint Eq. (35): $\widehat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m,o',j'} \geq \widehat{c}_{r-1,m}$
    end

    ---

    if $\{r = 1$ **and** $o > 1\}$ **then**
      $m' =$ **Job**$[j]$.**Sublot**$[s]$.**Operation**$[o-1].MachineIndex$
      $r' =$ **Job**$[j]$.**Sublot**$[s]$.**Operation**$[o-1].RunIndex$
      ADD-Constraint Eq. (36): $\widehat{c}_{1,m} - b_{s,j} \cdot T_{o,j,m} - S^*_{o,j,m} \cdot A_{o,j} - \geq \widehat{c}_{r',m'} + L_{o,j}$
    end

    ---

    if $\{r > 1$ **and** $o > 1\}$ **then**
      $m' =$ **Job**$[j]$.**Sublot**$[s]$.**Operation**$[o-1].MachineIndex$
      $r' =$ **Job**$[j]$.**Sublot**$[s]$.**Operation**$[o-1].RunIndex$
      $j' =$ **Machine**$[m]$.**Run**$[r-1].JobIndex$
      $o' =$ **Machine**$[m]$.**Run**$[r-1].OperationIndex$
      ADD-Constraint Eq. (37): $\widehat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m,o',j'} \cdot A_{o,j} - \geq \widehat{c}_{r',m'} + L_{o,j}$
    end
  end
end

---

for $\{j \leftarrow 1$ **to** $J\}$ **for** $\{s \leftarrow 1$ **to** $S_j\}$ **do**
  ADD-Constraint Eq. (38): $b_{s,j} \leq B_j \times \gamma_{s,j}$
end

---

for $\{j \leftarrow 1$ **to** $J\}$ **do**
  ADD-Constraint Eq. (39): $\sum_{\forall s|(\gamma_{s,j}=1)} b_{s,j} = B_j$
end