Simultaneous balancing, sequencing, and workstation planning for a mixed model manual assembly line using hybrid genetic algorithm

Published in 2018 in Computers & Industrial Engineering, Vol. 119, 370-387

Please cite this article as:

Defersha, F. M., and Mohebalizadehgashti, F. (2018) Simultaneous balancing, sequencing, and workstation planning for a mixed model manual assembly line using hybrid genetic algorithm, Computers & Industrial Engineering, Vol. 119, pp.370-387

Simultaneous Balancing, Sequencing, and Workstation Planning for a Mixed Model Manual Assembly Line using Hybrid Genetic Algorithm

Fantahun M. Defersha*, Fatemeh Mohebalizadehgashti

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

Abstract

Balancing and sequencing are two important challenging problems in designing mixed-model assembly lines. A large number of studies have addressed these two problems both independently and simultaneously. However, several important aspects such as assignment of common tasks between models to different workstations, and minimizing the number and length of workstations are not addressed in an integrated manner. In this paper, we proposed a mixed integer linear programming mathematical model by considering the above aspects simultaneously for a continuously moving conveyor. The objective function of the model is to minimize the length and number of workstations, costs of workstations and task duplications. Since the proposed model cannot be efficiently solved using commercially available packages, a multi-phased linear programming embedded genetic algorithm is developed. In the proposed algorithm, binary variables are determined using genetic search whereas continuous variables corresponding to the binary variables are determined by solving linear programming sub-problem using simplex algorithm. Several numerical examples with different sizes are presented to illustrate features of the proposed model and computational efficiency of the proposed hybrid genetic algorithm. A comparative study of genetic algorithm and simulated annealing is also conducted.

Keywords: Balancing; Sequencing; Mixed Integer Linear Programming Model; Mixed Model Assembly Line; Hybrid Genetic Algorithm.

1. Introduction

Assembly lines are types of manufacturing systems in which products are progressively assembled along a sequence of workstations. They are generally classified as single-model, mixed-model, and multi-product assembly lines. Single-model assembly line is the simplest

*corresponding author

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

of all, and as its name implies only one model of a given product is assembled. Whereas, in mixed-model situation, different models of a product are assembled one after the other without forming batches of identical models and without requiring setup between different models. In multi-product assembly lines, relatively different products are assembled in batches where one batch of a product is followed by a batch of another product with a significant setup time. Among these three assembly line types, mixed-model assembly line is widely studied and used in industry as it enables companies to produce different models of one product simultaneously to satisfy varying needs of customers in a responsive manner. Differences of models come from various factors such as size and color diversity, applied materials or even equipment. Therefore, varying assembly tasks, different task times and precedence relations are required to produce them (Becker and Scholl, 2006).

Several issues should be considered in designing a mixed-model assembly line. These include line balancing, layout design, and model sequencing (Manavizadeh et al., 2012; Ho, 2005; Boysen et al., 2009). The balancing and model sequencing are the main challenges for the planners of the mixed-model assembly line (McMullen and Frazier, 2000). The former requires assigning tasks to different workstations as evenly as possible while satisfying various constraints, such as the precedence relations among task and cycle time constraint (Simaria and Vilarinho, 2004). The sequencing problem, on the other hand, focuses on determining the sequence of the different models while meeting model mix requirements and minimizing line starvation and congestion (Scholl et al., 1998). These problems are generally considered hierarchically. The hierarchical manner focuses on balancing the assembly line first. Following that, the sequencing problem is solved (Mosadegh et al., 2012a). The challenge of balancing and sequencing can be more amplified in designing assembly lines with continuously moving conveyors. When the conveyor of the assembly line is moving continuously (opposed to intermittent synchronous motion), not only the number of workstation but also the length of workstations, the starting and finishing location of each task on the conveyor need to be determined. In this paper, we consider balancing and sequencing problems simultaneously assuming a continuously moving conveyor. The remainder of this paper is organized as follows: Section 2 provides a literature review. In Section 3, the proposed mixed integer linear programming model ($MILP$) is presented. A solution procedure based on genetic algorithm is presented in Section 4. Several numerical examples are presented in Section 5 to illustrate the problem addressed in this paper and show the computational efficiency of the proposed algorithm. Finally, conclusions are given in Section 6.

2. Literature Review

Mixed-model assembly line balancing and sequencing problems have been widely studied in literature. Comprehensive surveys of many of these studies can be found in Becker and Scholl (2006) and Boysen et al. (2007). Vilarinho and Simaria (2006) employed ant colony algorithm

to solve a balancing problem with parallel workstations and zoning constraints. Yagmahan (2011) solved mixed-model assembly line balancing problem by proposing a multi-objective ant colony optimization algorithm. A mixed integer linear programming model in the presence of parallel workstations, zoning constraints, and sequence-dependent set-up times between tasks was proposed in Akpinar and Baykasoglu (2014). The authors employed a multiple colony hybrid bees algorithm to solve the proposed model. Rabbani et al. (2016a) proposed a multi-objective model and evolutionary algorithms to solve balancing problem of a U-shaped mixed-model assembly line with the focus on minimizing the cycle time and the number of workstations, and maximizing the line efficiencies. Kucukkoc and Zhang (2016b) developed ant colony optimization algorithm to solve balancing problem in a mixed-model parallel two-sided line. Roshani et al. (2017) proposed a mathematical model and simulated annealing algorithm to solve balancing problem of an assembly line with multi-manned workstations. The objectives of the proposed model were: minimizing the total number of workers on the line and minimizing the number of multi-manned workstations. Rabbani et al. (2016b) proposed a multi-objective model and algorithms to solve balancing in the mixed-model assembly line with parallel workstations in a dynamic situation.

The papers reviewed above are mainly concerned with line balancing. Numerous studies were also conducted to solve sequencing problem. A comprehensive review of many of these studies was conducted by Boysen et al. (2009). Ishigaki and Miyashita (2016) used simulated annealing algorithm to solve sequencing problem. Makarouni et al. (2016) developed an integer programming model with the objective to maximize the just-in-time use of resources by minimizing the differences between actual and planned production dates. A greedy randomized adaptive search procedure (GRASP) was developed in Bautista et al. (2016) for a sequencing problem with the focus on minimizing work overload and unused assembly time. Bautista et al. (2017) proposed a hybrid meta-heuristic by combining dynamic programming and linear programming. The objective of their study was to minimize the total work overload. Guo and Ryan (2017) proposed a stochastic mixed-integer model to minimize the total earliness and lateness when the finished products have due dates.

Many research articles that attempt to solve balancing and sequencing problems in a hierarchical manner have also been published. For example, Sawik (2002) proposed a monolithic and a hierarchical approach to solve balancing and sequencing problems of a flexible assembly line. The author developed mixed integer programming models to minimize the completion time of products. Hwang and Katayama (2010) solved balancing and sequencing problems in a hierarchical manner to minimize the number of workstations and the variance of their workload. Faccio et al. (2016) solved two problems of the paced mixed-model assembly line hierarchically with using a supplementary flexible operators, so-called jolly operators. Objectives of their study were to minimize the number of jolly operators and work-overloads. Fish School Search algorithm (FSSA) was proposed in Monteiro Filho et al. (2017) to solve

balancing and sequencing problems hierarchically. The results were compared with Particle Swarm Optimization algorithm (PSO). PSO outperformed FSSA in solving balancing problem. However, FSSA gave more efficient results in solving the sequencing problem.

The articles reviewed so far address either balancing or sequencing problem or both problems hierarchically. There are also a considerable number of studies carried out to solve these problems simultaneously. Kim et al. (2000) solved balancing and sequencing problems simultaneously by employing a genetic algorithm. Their study aimed to minimize the total utility work, which is the total amount of work that is not completed within the given length of a workstation. Bock et al. (2006) proposed a new mathematical model and a simulated annealing based solution procedure. The objective is to minimize the total cost related to wages for the operators, overtime, wages for the floaters (operators assigned temporarily to a workstation), and for off-line repair if a work overload does not allow the correct production of a specific product. Saif et al. (2014) utilized a multi-objective artificial bee colony algorithm to minimize the total flow time of models, decreasing the workload deviations of stations from the average workloads, and reducing the number of incomplete units by balancing the workload on each station. Manavizadeh et al. (2015) proposed a multi-objective model and a heuristic algorithm to simultaneously solve the balancing and sequencing problems in the U-shaped assembly line. Kucukkoc and Zhang (2016a) developed a hybrid algorithm, which was a combination of genetic algorithm and ant colony optimization algorithm to solve balancing and sequencing problems in a parallel two-sided assembly line. Overall, the studies mentioned above have made notable contributions towards developing models for solving balancing and sequencing problems in mixed-model assembly lines. Nonetheless, they did not consider several important aspects such as assigning common tasks between various models of a product to different workstations, which is called task duplication. Ignoring this aspect without considering the related costs can reduce the number of feasible and efficient configurations (Bukchin and Rabinowitch, 2006). In addition, the above studies did not attempt to minimize the number and the length of workstations in an integrated manner. In this paper, we developed a mathematical model that incorporates many of the above aspects of assembly lines by assuming a continuously moving conveyor. A multi-phased linear programming embedded genetic algorithm is also developed to effectively solve the proposed model.

3. Mathematical Model

3.1. Problem definition:

Assume a mixed-model assembly line intended to manufacture a total of M different models of a product. The demand D_m for each model in a given time period is also assumed to be known and hence the model launching interval (L_r cycle time) is determined. In this scenario, the demand can be broken into f cycles in order to use a cyclic production strategy where f is the greatest common divisor of demand values. The vector $d = d_1, \ \ldots, \ d_m$, where

$d_m = \frac{D_m}{f}$, represents the model mix called Minimum Part Set (MPS) to be manufactured in each production cycle (Hyun et al., 1998; Mosadegh et al., 2012a). A total of f repetitions of the production of the MPS is required to satisfy the demands for the models. Figure 1 illustrates a typical assembly line producing two models (A and C) with $d_1 = d_A = 1$ and $d_2 = d_C = 2$. Given the precedence relationship of the tasks for each model, the demand vector $d = d_1, \ \ldots, \ d_m$, the launching interval L_r, and the conveyor speed, the problem is to determine (1) the sequence of model launching, (2) the assignment of tasks to the various station and (3) the starting and finish location of the tasks on the conveyor. The objective is to minimize weighted sum of the number of workstations, the length of the assembly line and the cost of task duplication. A typical solution resembles the one given in Figure 1. The thick solid line (XX) represents the conveyor of the assembly line. The Gantt chart indicates the starting and finish locations of the tasks along the conveyor. The length of the assembly line is the sum of the lengths of all the stations created ($w_1 + w_2 + ... + w_n$). Task duplication cost occurs when a common task of models is assigned to different stations. In Figure 1, for example, Task-5 of Model A is assigned to station-1 while this task in Model C is assigned to Station-3. Such task duplication may be needed to better balance the workstations and reduce the length of the line.

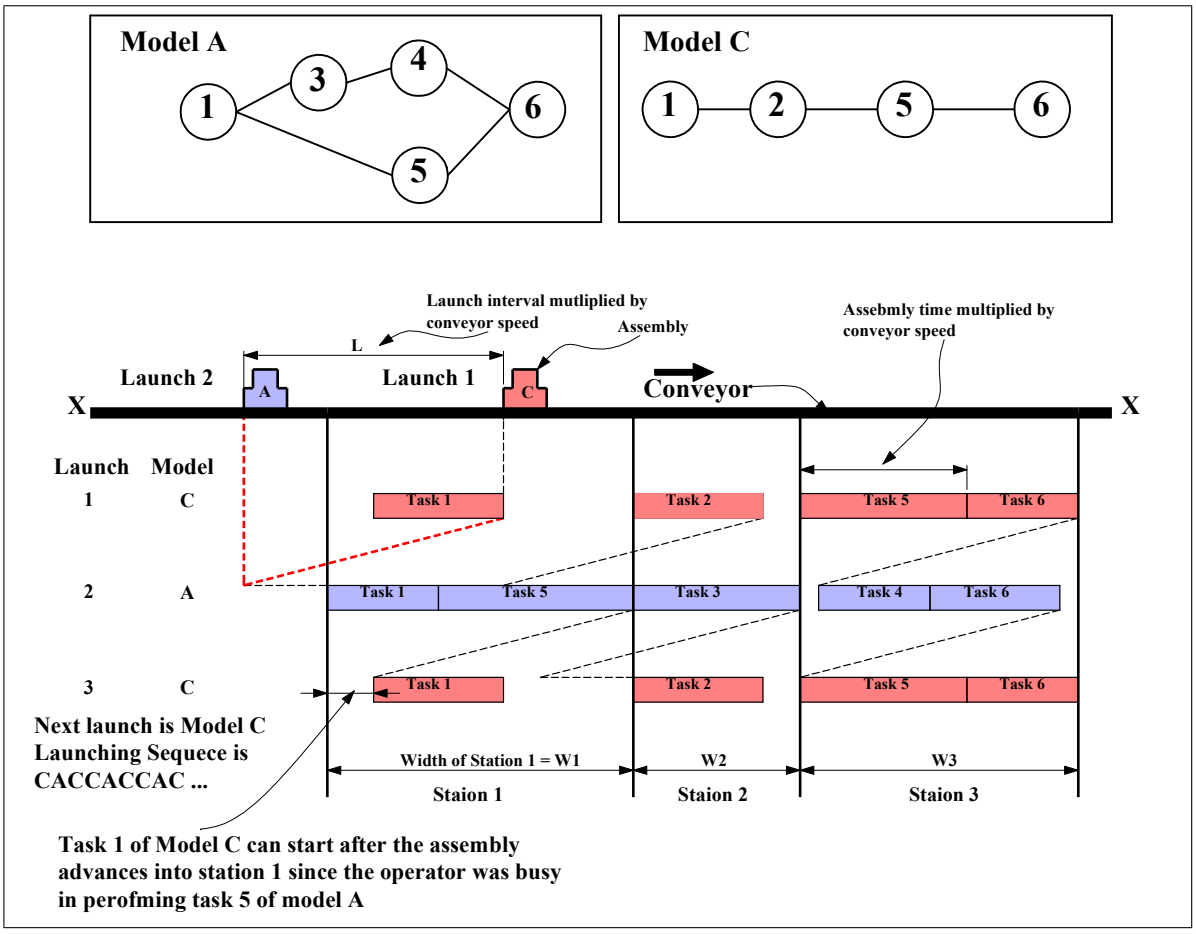


Figure 1: Example precedence diagrams and solution for two models A and C

3.2. Problem formulation:

In this section, a mixed-integer linear programming ($MILP$) model is proposed based on the study conducted by Mosadegh et al. (2012a) where the author assumes the number and length of workstations are given. In our mathematical model, these are decision variables. The following list provides the indices, parameters, and variables of the proposed model.

Indices:

m	Index for model
$t,\ h$	Index for tasks
s	Index for sequence
k	Index for station

Parameters:

M	Total number of models where models are indexed by $m = 1,\ \ldots,\ M$
T	Total number of tasks where tasks are indexed by t or h where $t = h = 1,\ \ldots,\ T$
S	Total number of sequences where sequences are indexed by $s = 1,\ \ldots,\ S$
K	Maximum number of workstations where workstations are indexed by $k = 1,\ \ldots,\ K$
v	Speed of conveyor
L	Launching rate of each model
$Pre_{m,t}$	Set of immediate precedent tasks for task t of model m
$A_{m,t}$	Assembly time for task t of model m (Note: $A_{m,t} = 0$ if task t is not needed in model m).
$\tilde{A}_{m,t}$	Binary data which equals to 1 if $A_{m,t} \geq 0$, 0 otherwise.
d_m	Demand of model m in the MPS
SC	Station cost, fixed cost associated with each workstation
TC	Task duplication cost
B	A large positive number

Variables:

Continuous Variables:

w_k	Length of workstation k
$p_{s,k}$	Start position of operator at sequence s in workstation k

Binary Variables:

$x_{m,t,s,k}$ Binary variable which equals to 1 if task t of model m at sequence s is assigned to workstation k, 0 otherwise

$a_{t,k}$ Binary variable which equals to 1 if task t of any model is assigned to workstation k, , 0 otherwise

z_k Binary variable which equals to 1 if workstation k is open, 0 otherwise

$y_{m,s}$ Binary variable which equals to 1 if sequence s is assigned to model m, 0 otherwise

---

Minimize:

$$Objective = f_1 \cdot \sum_{k=1}^{K} w_k + f_2 \cdot \sum_{k=1}^{K} SC \cdot z_k + f_3 \cdot \sum_{t=1}^{T} TC \cdot ((\sum_{k=1}^{K} a_{t,k}) - 1) \tag{1}$$

Subject to:

$$p_{s,k} + (\sum_{m=1}^{M} \sum_{t=1}^{T} A_{m,t} \cdot x_{m,t,s,k}) \cdot v \leq w_k \ ; \quad \forall (s,k) \tag{2}$$

$$p_{s,k} + (\sum_{m=1}^{M} \sum_{t=1}^{T} A_{m,t} \cdot x_{m,t,s,k}) \cdot v - L \cdot v \leq p_{s+1,k} \ ; \quad \forall (s,k) | (s < S) \tag{3}$$

$$p_{S,k} + (\sum_{m=1}^{M} \sum_{t=1}^{T} A_{m,t} \cdot x_{m,t,S,k}) \cdot v - L \cdot v \leq p_{1,k} \ ; \quad \forall (k) \tag{4}$$

$$\sum_{k=1}^{K} x_{m,t,s,k} = y_{m,s} \cdot \tilde{A}_{m,t} \ ; \quad \forall (m,t,s) \tag{5}$$

$$\sum_{s=1}^{S} y_{m,s} = d_m \ ; \quad \forall (m) \tag{6}$$

$$\sum_{m=1}^{M} y_{m,s} = 1 \ ; \quad \forall (s) \tag{7}$$

$$\sum_{k=1}^{K} k \cdot x_{m,h,s,k} \leq \sum_{k=1}^{K} k \cdot x_{m,t,s,k} \ ; \quad \forall (m,t,s,h) | h \in Pre_{m,t} \tag{8}$$

$$w_k \leq B \cdot z_k \ ; \quad \forall (k) \tag{9}$$

$$x_{m,t,s,k} \leq z_k \ ; \quad \forall (m,t,s,k) \tag{10}$$

$$z_k \geq z_{k+1} \ ; \quad \forall (k) \tag{11}$$

$$a_{t,k} \geq x_{m,t,s,k} \; ; \quad \forall (m,t,s,k) \tag{12}$$

$$x_{m,t,s,k},\ y_{m,s},\ a_{t,k} \text{ and } z_k \text{ are binary} \tag{13}$$

$$p_{s,k} \text{ and } w_k \text{ are greater than equal zero} \tag{14}$$

The objective function of the model in Eq. (1) minimizes the number and length of workstations and also the total cost, which is the sum of workstations cost and tasks duplication cost. The constraint set in Eq. (2), declares that all operations should be performed within the length of the workstation. The constraint given in Eq. (3) determines the starting position of the operator in the workstation after finishing each task of each model in each sequence (except for the last sequence). The constraint given in Eq. (4) determines the start position of operator in the last sequence of each cycle. The constraint in Eq. (5) states that each task of each model is assigned to a particular sequence, if its model is assigned to that sequence before. Eq. (6) emphasizes that the demand for each model in the MPS must be satisfied. Eq. (7) guarantees that each model is assigned to a specific sequence. Therefore, all tasks of a special model are completed in the same sequence. Precedence constraints are satisfied in the constraint set of Eq. (8). The constraint set in Eq. (9) declares that the workstation length will be zero if that workstation is not open in the assembly line. Eqs. (10) and (11) and Eq. (12) impose the logical constraints on the binary variables. Specifically, the constraint set in Eq. (10) states that a task is assigned to a particular workstation if that workstation is open. Eq. (11) prevents a gap between two consecutive opened workstations in the assembly line. Constraint set of Eq. (12) declares that similar tasks of different models can be assigned to different workstations. Eqs. (13) and (14) shows binary variables and variables that are greater than equal zero respectively.

4. Hybrid Genetic Algorithm

Assembly line balancing and sequencing problems are individually NP-hard (Mosadegh et al., 2012b). In light of this fact, we can infer that the proposed model which combines both problems is also NP-hard. To efficiently solve this model, we developed a linear programming embedded hybrid genetic algorithm (HGA). The genetic algorithm searches over the integer variables. For each integer solution visited, a simplex algorithm is used to solve a linear programming sub-problem in order to determine the continuous variables that optimally correspond to a given integer solution. The following sections provide the components of the developed algorithm.

4.1. Chromosomal encoding and decoding

Solution representation is the first and the most important step in applying a metaheuristic. In this paper, we develop a solution representation that encodes only the integer variable in such a way that a randomly generated solution can satisfy the constraints of the model that are composed of only the integer variable. The continuous variables and the constraints that contain them are being taken care by solving a linear programming sub-problem. Moreover, the solution representation is designed to enable the search to be executed in three consecutive phases. In the first phase, the search is limited to finding solutions that do not allow task duplication. In the second phase, the search is expanded to include solutions that allow task duplication across different models. The complete search space of the model that includes solutions that allow task duplication not only across models but also within the different occurrences of the same model in the sequence is explored in the third phase. Figure 2 presents the general structure of the solution representation. An example of this structure is depicted in Figure 3 assuming three models ((A, B, and C) or (1, 2, and 3)) with MPS $[d_1 = 2, d_2 = 1, \text{and } d_3 = 2]$. The tasks superscripted with an asterisk (*) in some segments are just placeholders as they are not required in the respective models. For example, task 6 in segment-6 is just a placeholder as this task is not needed in model A. It can appear on any place in this segment without precedence requirement and can be assigned to a station with zero processing time. The different components of the solution representation are explained in details in the following subsections.

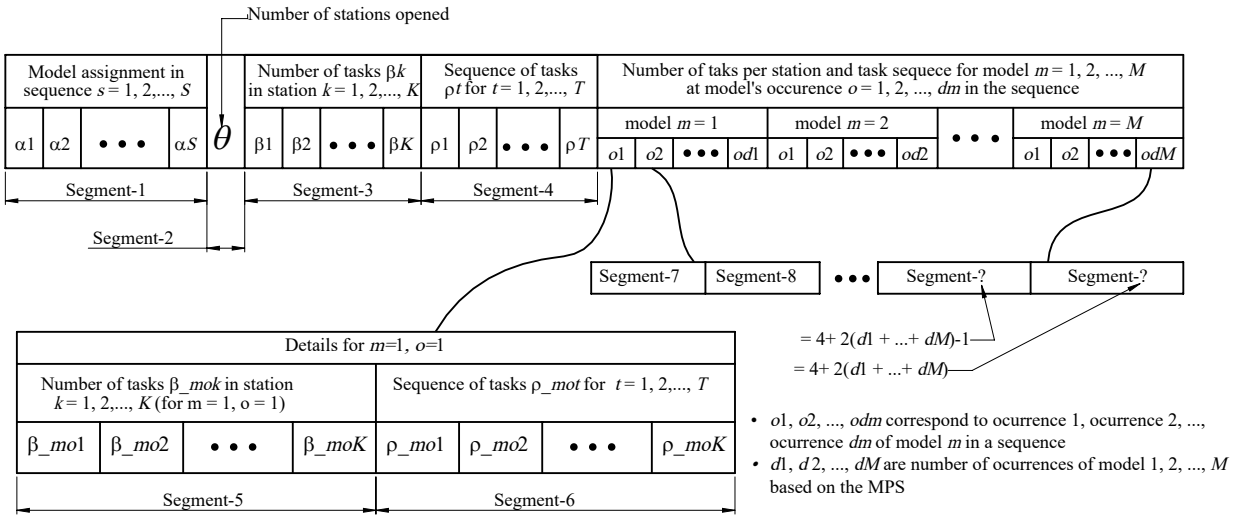


Figure 2: Solution Representation

4.1.1. Segment-1

Segment-1 of the solution representation (used in all the three phases of the search) determines the sequence of the models. In this segment, the gene α_s takes an index of a model. Its

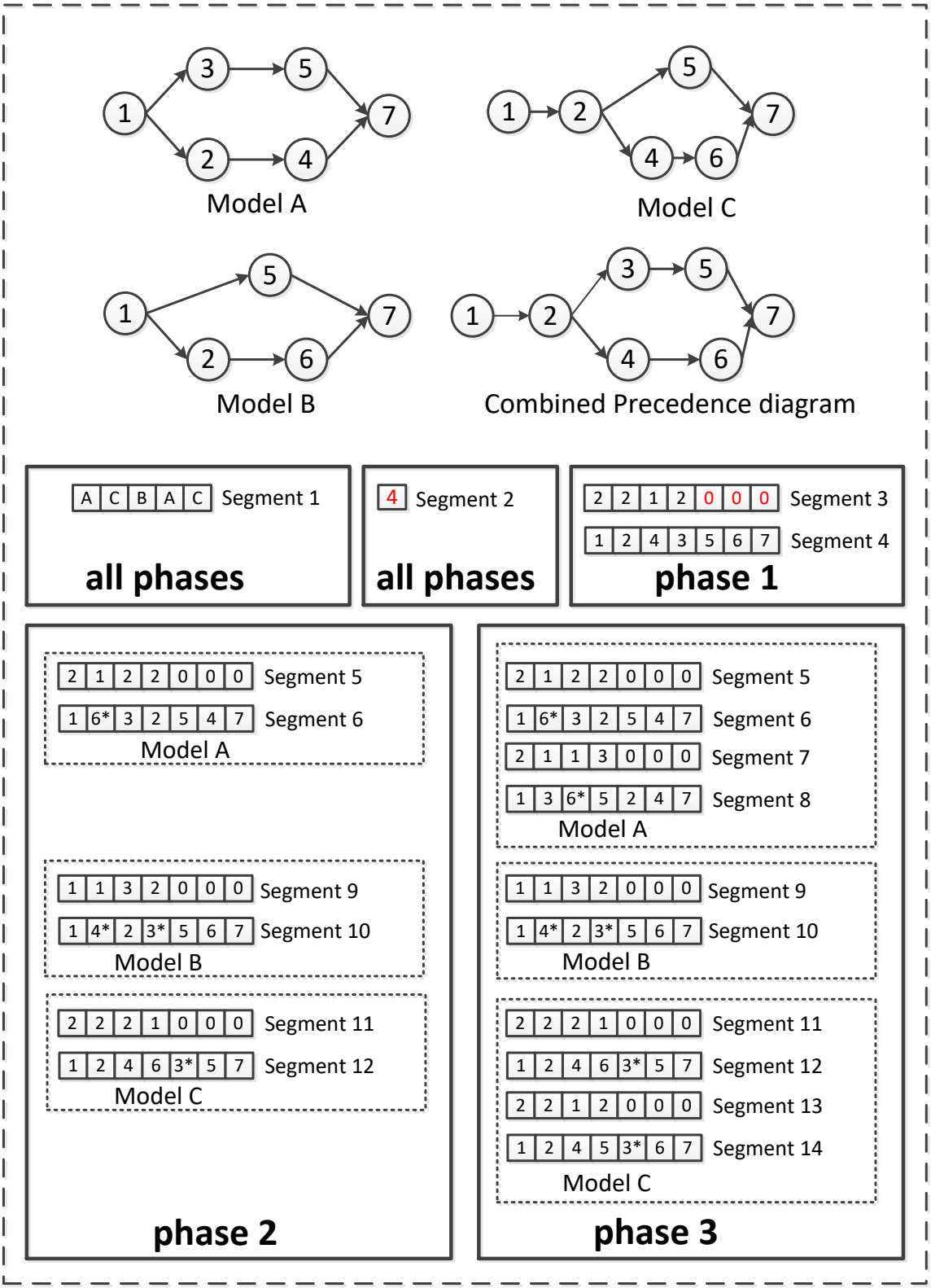


Figure 3: A typical precedence diagram and parts of a solution representation applicable during different phases of the search (assuming $d_A = 2$, $d_B = 1$, and $d_C = 2$ and $\theta = 4$ in Segment-2; the tasks superscripted by asterisk * in some segments are just placeholders as they are not required in the respective models).

length S is equal to the sum of demands of different models in the MPS. In Figure 2, this segment indicates the first launch is model A, followed by C where the complete sequence is [A C B A C]. The variable $y_{m,s}$ can be decoded from this segment using Eq. (15), satisfying constraints in Eqs. (6) and (7) of the model.

$$y_{m,s} = \begin{cases} 1 & ; \text{if } \alpha_S = m \\ 0 & ; \text{otherwise} \end{cases} \tag{15}$$

4.1.2. Segment-2

This segment has a single gene θ which takes a value in $\{1, K\}$ to denote the number of stations opened where K is the maximum number of workstations that can be opened (theoretically $K = T$). The binary variable z_k is decoded from this segment using Eq. (16) satisfying constraint Eq. (11). In the illustrative example in Figure 3, we assume $\theta = 4$. This segment is used in all the three phases of the search.

$$z_k = \begin{cases} 1 & k = 1, 2, \ \ldots, \ \theta \\ 0 & k > \theta \end{cases} \tag{16}$$

4.1.3. Segment pair 3-4

These two segments are used only in the first phase of the algorithm. The gene β_k in Segment-3 indicates the number of tasks assigned to station k. Given the number of opened stations θ in Segment-2, the values of β_k should satisfy the conditions in Eqs. (17) and (18) where T is the total number of tasks. In Figure 3, this segment has the values [2 2 1 2 0 0 0]. The gene ρ_t in Segment-4 (for $t = 1, \ 2, \ \cdots, \ T$ and $\rho_t \neq \rho_{t'}$) takes the index of one of the tasks in such a way that the segment represents a subset of the permutations of the tasks that can satisfy the precedence requirement in all the models. In the example shown in Figure 3, the vector of these genes has the value [1 2 4 3 5 6 7]. By using the information in Segment-3 and Segment-4, the tasks can be assigned to the various stations. For example, from Segment-3, the number of tasks assigned to Station-1 is 2. Thus, the first two tasks (tasks 1 and 2) in Segment-4 are to be assigned to Station-1. Following this procedure, the task-station assignments are as follows: tasks 4 and 3 to station 2, task 5 to station 3, and tasks 6 and 7 to station 4. This assignment of the tasks to stations will provide the values of the decision variables $x_{m,t,s,k}$ and $a_{t,k}$. The assignment of the tasks to various stations in this first phase of the search is the same for all models in the sequence.

$$\beta_1 + \beta_2 + \ldots + \beta_\theta = T \tag{17}$$

$$\beta_{\theta+1} + \beta_{\theta+2} + \ldots + \beta_K = 0 \tag{18}$$

4.1.4. Segment pairs 5-6, 9-10, and 11-12

In the second phase of the search, the task-assignment to various stations is done for each model independently by keeping the same task-assignment of a given model in all of its occurrences in the sequence. In this case, each model requires two pairs of segments in the solution representation. For instance, segment pairs 5-6 can be used to assign the tasks of both of the two occurrences of model A in the sequence (note that A occurs twice in Segment-1). The pairs 9-10 and 11-12 are for models B and C, respectively. Task-station assignments are done in a similar way as discussed previously, though performed for each model separately.

4.1.5. Other segment

In the third phase of the search, the task-assignment to the various stations is done for each occurrence of a model independently. For instance, model A occurs two times in the sequence. Thus, segment pairs 5-6 and 7-8 are used for task-station assignment of the first and second occurrence, respectively. Model B has only a single occurrence and hence only one pair of segments (namely 9-10) is sufficient both in the second and third phase. Model C has two occurrences requiring two pairs of segments (11-12 and 13-14).

4.2. Linear programming subproblem

The values of the binary variables for the $MILP$ model in section 3.2 are obtained by decoding the chromosome through task-station assignment discussed above. A close observation of this decoding procedure can reveal that the constraints of the model that are composed of only integer variables are satisfied. The continues variables that optimally correspond to an integer solution are determined by formulating and solving a linear programming subproblem. The linear programming subproblem is formulated by selectively removing the constraints and the objective function terms of the original MILP model that are composed of only the integer variable. The resulting LP model is given below.

Minimize:

$$Objective = f_1 \cdot \sum_{k=1}^{K} w_k \tag{19}$$

Subject to:

$$p_{s,k} + (\sum_{m} \sum_{t} A_{m,t}) \cdot v \leq w_k \ ; \quad \forall (s,k) \tag{20}$$

$$p_{s,k} + (\sum_{m} \sum_{t} A_{m,t}) \cdot v - Lr \cdot v \leq p_{s+1,k} \ ; \quad \forall (s,k) | (s < S) \tag{21}$$

$$p_{S,k} + (\sum_{m} \sum_{t} A_{m,t}) \cdot v - Lr \cdot v \leq p_{1,K} \ ; \quad \forall (k) \tag{22}$$

$$w_k \leq B \ ; \quad \forall (k) \tag{23}$$

4.3. Fitness function

The objective function of the model in Eq. (1) is used as a fitness function to evaluate a particular solution. This function is evaluated as follows. First, the integer variables are decoded from a solution under evaluation. Using the integer solution, the second and third terms of the objective function are determined. Second, the LP-subproblem corresponding to the integer solution is solved to provide the value of the first term of Eq. (1). The sum of the three terms provides the fitness of the solution under evaluation. Since the objective function is minimization, a particular solution is termed as fit if it has a small value of its fitness.

4.4. Genetic operators

In a genetic algorithm, operators are required to evolve a population of solutions towards a promising region of the search space. Generally, these operators are classified as selection, crossover, and mutation operators. The following subsections provide details of these operators as applied to the proposed algorithm.

4.4.1. Selection operators

Selection in a GA is a step in which individual chromosomes are chosen from a population based on their fitness values. The more fit a chromosome is the more copies of this chromosome added to the breeding pool to form the next generation of solutions. This operator can be implemented in different ways. Roulette wheel and tournament based selections are commonly used in literature. In this paper, the tournament selection method is employed. In tournament selection, first a set of k individuals are randomly selected from the population; then, one individual with the smallest fitness (for minimization problem) is selected as the best one and its copy is added to the mating pool. This procedure is repeated with replacement (previously selected individuals are also computing) until the number of individuals in the mating pool reaches the required population size.

4.4.2. Crossover operators

Once the breeding pool is formed by the selection operator, individuals in the pool are randomly paired. On each pair, different crossover operators are applied with certain probabilities to generate offsprings. In the proposed algorithm, we develop nine crossover operators, namely XO1, XO2, ..., XO8 whose functions are detailed below. They are applicable in different phases of the search. XO1 is applicable in all the three phases. Operators XO2 and XO3 are applicable only in phase-1 whereas XO4 and XO5 are only in phase-2. The operators XO6, XO7 and XO9 are applicable only in the third phase of the search. The actions of the crossover operators are described below.

XO1 Exchanges segment-1 between parent chromosomes.

XO2 Exchange segment-2 and -3 of the first parent chromosome with the corresponding segments of other parent chromosome.

XO3 Exchanges segment-4 between parent chromosomes.

XO4 Under this operator, segments-2 and all the other segments that determine the number of tasks on each station corresponding to all the first occurrences of all models [(m=1, o1), (m=2, o1), (m=2, o1), ..., (m=M, o1)] (see Figure 2) from one parent are exchanged with that of the other parent. For the example in Figure 3, this means that segments 2, 5, 9, and 10 of one parent are exchanged with that of the other parent.

XO5 Under this operator, segments provide the sequence of takes corresponding to all the first occurrences of all models [(m=1, o1), (m=2, o1), (m=2, o1), ..., (m=M, o1)] from one parent are exchanged with the other parent. For the example given in Figure 3, this means that segments 6, 10, and 12 of one parent are exchanged with that of the other parent.

XO6 Under this operator, segments-2 and segments determining the number of task on each station corresponding to all the occurrences of all models [(m=1, o1, o2, ..., od1), (m=2, o1, o2, ..., od2), ..., (m=M, o1, o2, ..., odM)] from one parent are exchanged with that of the other parent. In Figure 3, this means that segments 2, 5, 7, 9, 10, 13 of one parent are exchanged with that of the other parent.

XO7 This operator selects arbitrarily one model and exchanges the segments of the parent chromosomes that provide the sequence of the task for one of the occurrence of that model. In Figure 3, this means that only one of the segment out of 6, 8, 10, 12, 14 will be selected and exchanged between parent chromosomes.

XO8 This operator exchanges task-assignments in all occurrences of all the models between parent chromosome as long as the value of θ in segment-2 of the parents (the number of opened stations) are equal. In reference to Figure 3, this means that segments 5 to 14 of one parent will be exchanged with the other parent.

4.4.3. Mutation operators

A mutation operator acts with a small probability on a single chromosome to slightly alter its genetic makeup. In the proposed genetic algorithm, we developed ten Mutation Operators (MO1, MO2, ..., MO10) that are applicable during the different phases of the search. These operators are discussed in details below. (**Refer to Figure 3 for the examples given in the following discussion**).

MO1 This operator arbitrarily selects two genes in segment-1 of the chromosome and swap their values if they are different. E.g., [A C B A C] may be altered to [B C A A C] if the two different genes on the 1st and 3th locations are swapped (see Figure 3). This operator is applied in all the three phases of the search.

MO2 This operator steps up or down the gene value θ in segment-2 by one while adjusting the segment that determines the number of task on each station during phase-1 of

the search (i.e. segment-3). For example, if θ in segment-2 is changed from 4 to 3, segment-3 will be adjusted by raising the number of tasks to be assigned in each of the remaining three stations as equally as possible. Thus, the current values of segment-3, [2 2 1 2 0 0 0], before mutation, can be adjusted as [3 2 2 0 0 0 0] after mutation (there can be other possibilities). If θ is increased from 4 to 5, a fifth station will be created, and the number of tasks in this newly opened station will be raised from zero to the average number of tasks per station while reducing the number of tasks in each of the previous four stations as equally as possible. In this case, segment-3 can be adjusted to look like [2 1 1 1 2 0 0] after mutation.

MO3 This operator is similar to MO2, but it is applied in phase-2. Hence, whenever this operator is applied in segment-2, the segments that determine the number of task per station in each model need to be adjusted (these are segments 5, 9, and 11).

MO4 This operator is similar to MO2 and MO3, but it is applied in phase-3. Hence, whenever this operator is applied in segment-2, the segments that determine the number of task per station in each model and each occurrence need to be adjusted (these are segments 5, 7, 9, 11, and 13).

MO5 This operator arbitrarily selects two adjacent genes in Segment 3 (from those corresponding to opened stations only) during phase-1 of the search. Then, it arbitrarily selects one of these genes and step up its value by one and step down the value of the other gene by one (while preventing the values of the genes from being negative).

MO6 This operator is similar to MO5 but it is applied during Phase-2 of the search. It acts on one of the segments that determine the number of tasks on each station corresponding to the first occurrence of an arbitrarily selected model (In reference to Figure 3, either segment 5, 9 or 11 will be selected arbitrarily when this operator is applied).

MO7 This operator is similar to MO5 and MO6, but it is applied during Phase-3 of the search. It acts on one of the segments that determine the number of tasks on each station corresponding to an arbitrarily selected model and arbitrarily selected occurrence of that model (either segment 5, 7, 9, 11 or 13 will be selected arbitrarily when this operator is applied)

MO8 This operator arbitrarily selects a gene in segment-4 during phase-1 of the search and relocate it to a different location in this segment while obeying precedence requirement of tasks. E.g., if a gene with value 4 (task 4) is selected, it can be relocated either after the gene with value 3 or 5.

MO9 This operator is similar to MO8 but applied during phase-2 of the search. It acts on either segment 6, 10 or 12.

MO10 Similar to MO8 and MO9 but it is applied during phase-3 of the search. When it is applied, it acts on either segment 6, 8, 10, 12 or 14.

4.5. Flowchart of the hybrid genetic algorithm

Steps for applying the hybrid genetic algorithm are illustrated in a flowchart in Figure 4. These steps have been coded in C++. An ILOG-CPLEX modeling environment is used to solve the linear programming model using the simplex algorithm. The following notations are used in the flowchart:

p	Population size
c	Index for a chromosome
g	Generator counter
$maxg$	Maximum number of generations
$Phase$	An indicator number that takes number 1, 2, and 3 based on three defined phases
H	Number of iterations which generate populations successfully without any improvement in the best fitness function value so far obtained
H_{max1}	Maximum number of H that leads to entering to the second phase if the previous Phase was equal to 1
H_{max2}	Maximum number of H in the second phase that leads to entering to the third phase if the previous Phase was equal to 2
H_{max3}	Maximum number of H in the third phase that leads to stopping the third phase

5. Numerical Examples

In this section, we present four sets of numerical examples. The first numerical example is to illustrate the developed mathematical model. The second set of numerical examples is for the comparison of an off-the-shelf optimization package (ILOG-CPLEX) and the proposed genetic algorithm. The third set of numerical example is dedicated to comparing the performances of the proposed hybrid-genetic algorithm and against hybrid-simulated annealing. The hybrid simulated annealing was developed in this research for the purpose of comparison alone, and hence its detail implementation is not presented. The last and the forth numerical example is to demonstrate the robustness of the algorithm to its parameter settings using analysis of variance (ANOVA).

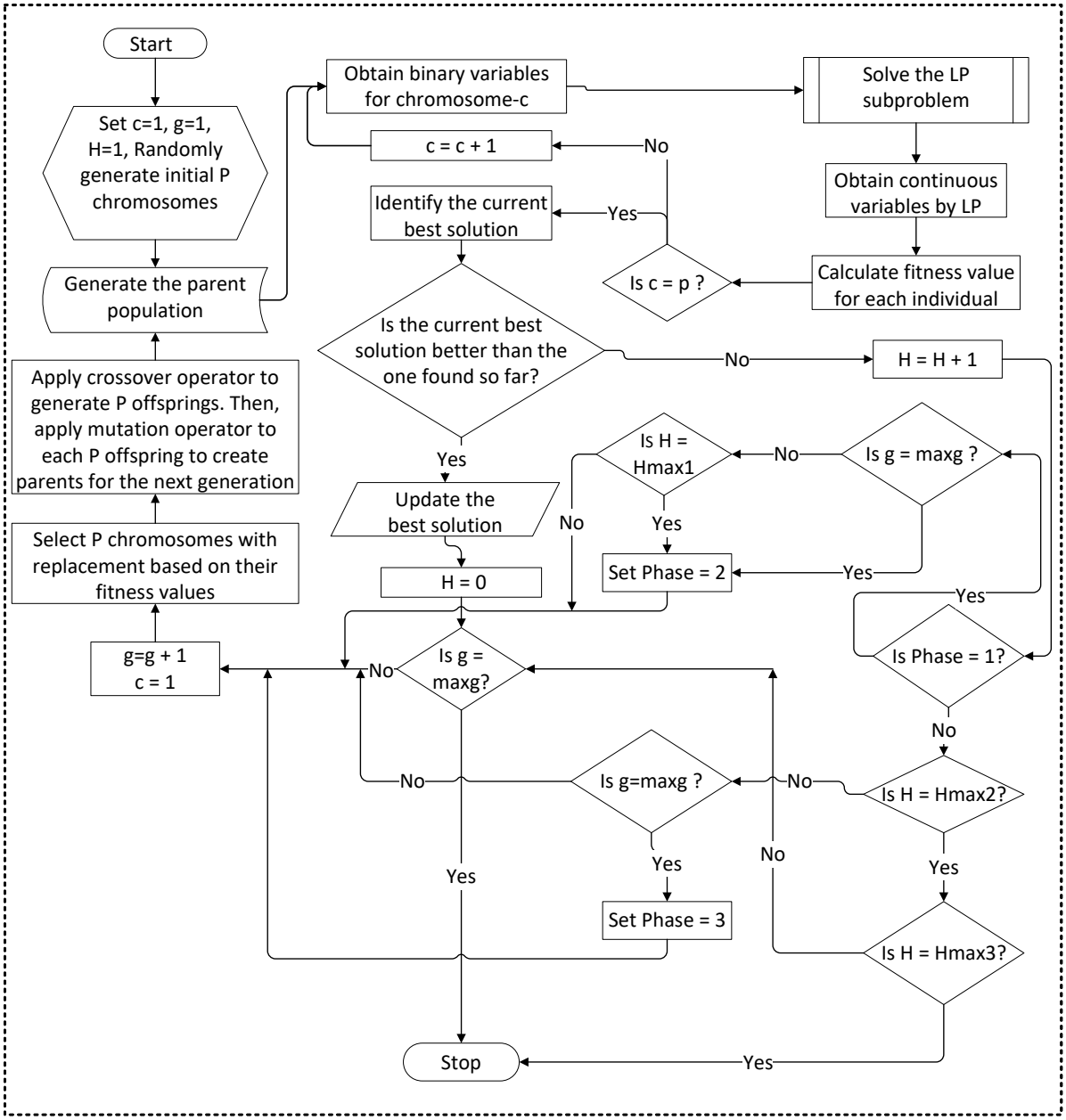


Figure 4: Flowchart of the hybrid genetic algorithm

5.1. Model Illustration

In this section, for a better comprehension of the mathematical model, we illustrate a typical solution using a small problem instance. The problem consists assembling three different models (A, B and C) of a product on continuously moving conveyor assembly line. Figure 5 provides the precedence diagram for the three models. The assembly time and duplication costs of the tasks are given in Table 1. For the purpose of this example, the following are assumed. An assembly has to be launched every 28 minutes, and a complete sequence is required to have 2 units of model A, 1 unit of model B and 3 units of model C. The cost of opening a station is 50 (in monetary units). The conveyor speed is 1.5 meters per minutes. The weight factors for the objective function terms are $f_1 = 5$, $f_2 = 1$ and $f_3 = 1$. The problem is solved to optimality using ILOG-CPLEX, and the resulting solution is depicted in Figure 6.

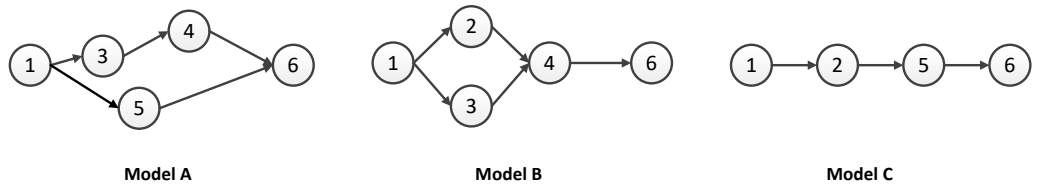


Figure 5: Precedence diagram for models A, B, and C for the problem used in model illustration

Table 1: Operation time of each task in each model and task duplication cost for the problem used in model illustration

	Tasks					
Models	1	2	3	4	5	6
A	12	0	18	12	21	14
B	10	16	22	10	0	16
C	14	14	0	0	18	22
Task duplication cost	10	5	7	9	4	8

In this figure, the thick dark line **XX** represent the conveyor. Task-1 of model C in the first launch can start only after its base part advances 7.5 meters into station 1. This is quite evident when we consider the first launch of the next cycle (after the 6th launch) where the operator was busy performing Task-5 of model A (launch 6) till the end of Station-1 at which time the base part of model C has already advanced 7.5 meters into this station. Once Task-1 of model C (first launch) is started, it advanced by 21 meters ($= A_{C,1} \times v$) before it gets completed. At this moment, its location is indicated as **Z1** on the Gantt chart. The actual location of this part on the conveyor is at **Z0**. The location of the next launch (model A) is indicated on the Gantt chart by drawing an included line from **Z1** to **Z2**. This line is drawn

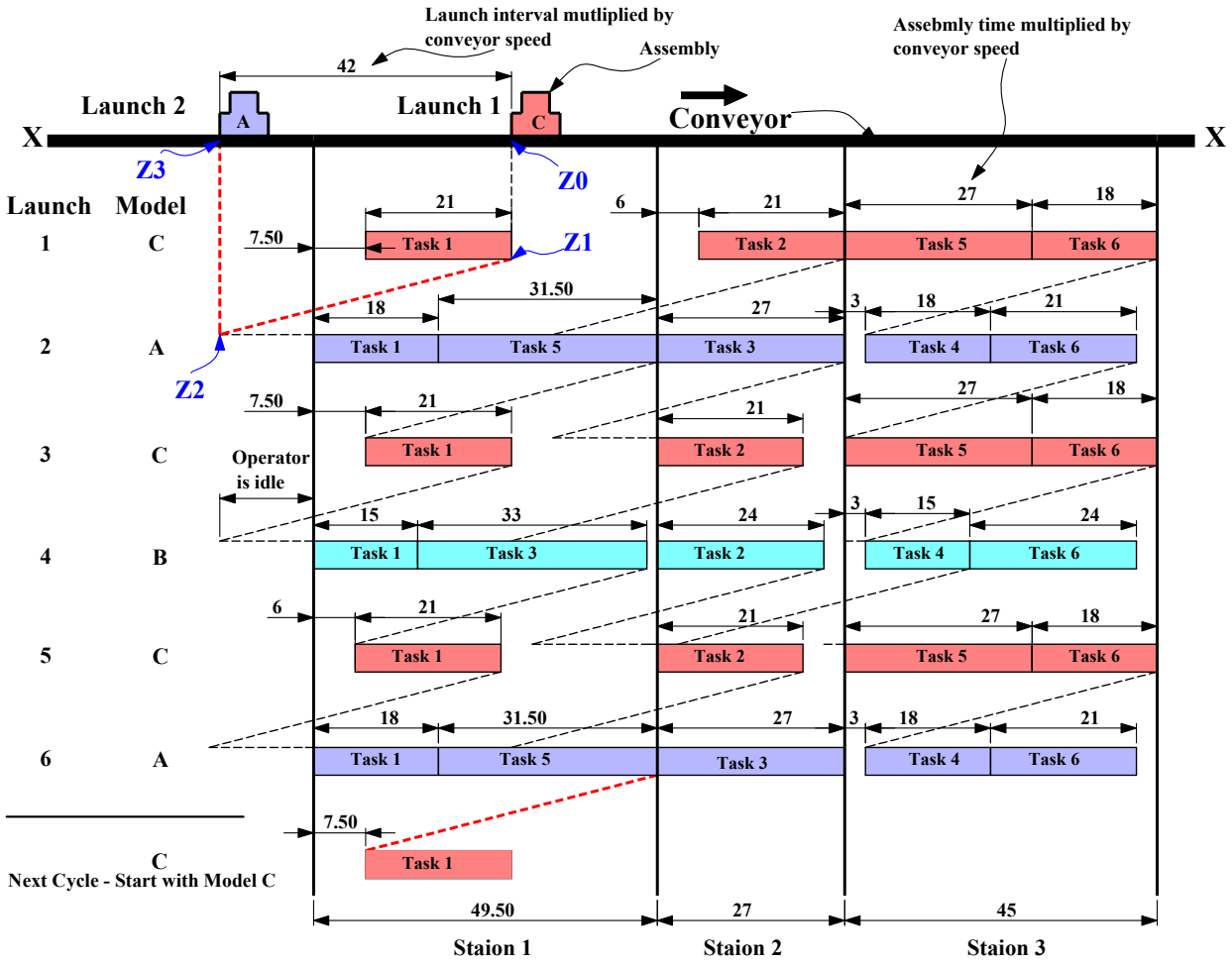


Figure 6: Optimal solution including task's assignment and model's sequence

in such a way that the actual location of the second launch on the conveyor **Z3** is at a distance of 42 meters $(L \cdot v)$ from the location of the first launch, **Z0**.

The value of the first term of objective function for this solution is $f_1 \cdot \sum_{k=1}^{K} w_k = 5 \times (49.50 + 27 + 45) = 607.5$. With three stations opened, the second term of the objective function $f_2 \cdot \sum_{k=1}^{K} SC \cdot z_k = 1 \times (50+50+50+0+0) = 150$. The value of the third term of the objective function is $f_3 \cdot \sum_{t=1}^{T} TC \cdot ((\sum_{k=1}^{K} a_{t,k}) - 1) = 7+4 = 11$. This value comes from task duplications. Tasks-3 of model A and model B are assigned to station 3 and 1, respectively, with a duplication cost of 7. Task 5 of model A is assigned to station 5 whereas this same task is assigned to station 5 in model C, incurring a duplication cost of 4 of this task. This brings the total task duplication cost to 11. A closer look into Figure 6 also shows that the solution satisfy all of the constraints of the model. For instance, the constraint in Eq. (2) for $s = 1$ and $k = 1$ is $p_{1,1} + (\sum_{m=1}^{M} \sum_{t=1}^{T} A_{m,t} \cdot x_{m,t,1,1}) \cdot v = 7.5 + (21) \leq w_1 = 45$. For $s = 1$ and $k = 1$, constraint Eq. (3) is $p_{1,1} + (\sum_{m=1}^{M} \sum_{t=1}^{T} A_{m,t} \cdot x_{m,t,1,1}) \cdot v - L \cdot v = 7.5 + (21) - 42 = -13.5 \leq p_{2,1} = 0$. The constraint in Eq. (4), for $k = 1$, is $p_{6,1} + (\sum_{m=1}^{M} \sum_{t=1}^{T} A_{m,t} \cdot x_{m,t,6,1}) \cdot v - L \cdot v = 0 + (18 + 31.5) - 42 = 7.5 \leq p_{1,1} = 7.5$.

5.2. HGA vs CPLEX

In this section, three problems with different sizes are considered to evaluate the computational efficiency and convergence behavior of the proposed hybrid genetic algorithm in comparison with the ILOG-CPLEX.

5.2.1. Problem-1

The problem considered in this example consists of three models with four tasks. Figure 7 illustrates the precedence diagram of each model. Table 2 provides information about the operation time (in time units) of each task in each model as well as the task duplication cost (in cost units). The launching rate, maximum number of stations and station cost are set at 20, 6, and 50, respectively. The speed of the conveyor is equal to 1 unit distance per unit time. For this problem, we considered two versions where the versions are differentiated by their MPS. In the first version, the demands of models in the MPS are $d_A = 2$, $d_B = 1$ 1 and $d_C = 2$ units. Therefore, there are five sequences in this version. For the first version of this example, the mathematical model presented in Section 3.2 has the total number of 471 variables, which 405 of these variables are integers. The total number of constraints is 940. In the second version, the number of demands for each model in the MPS is increased to $d_A = 3$, $d_B = 2$ and $d_C = 5$ units. This intern will increase the number of variable and constraints. This version of Problem-1 has the total number of 558 variables from which 480 are integers. The total number of constraints is 1119.

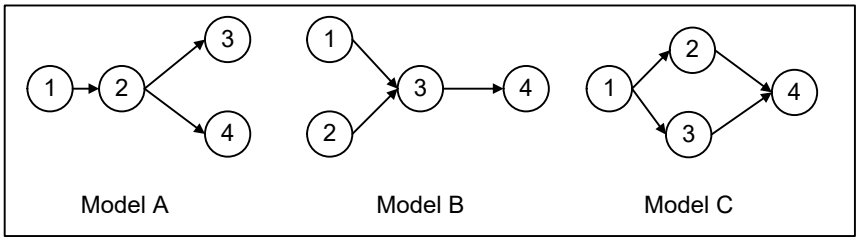


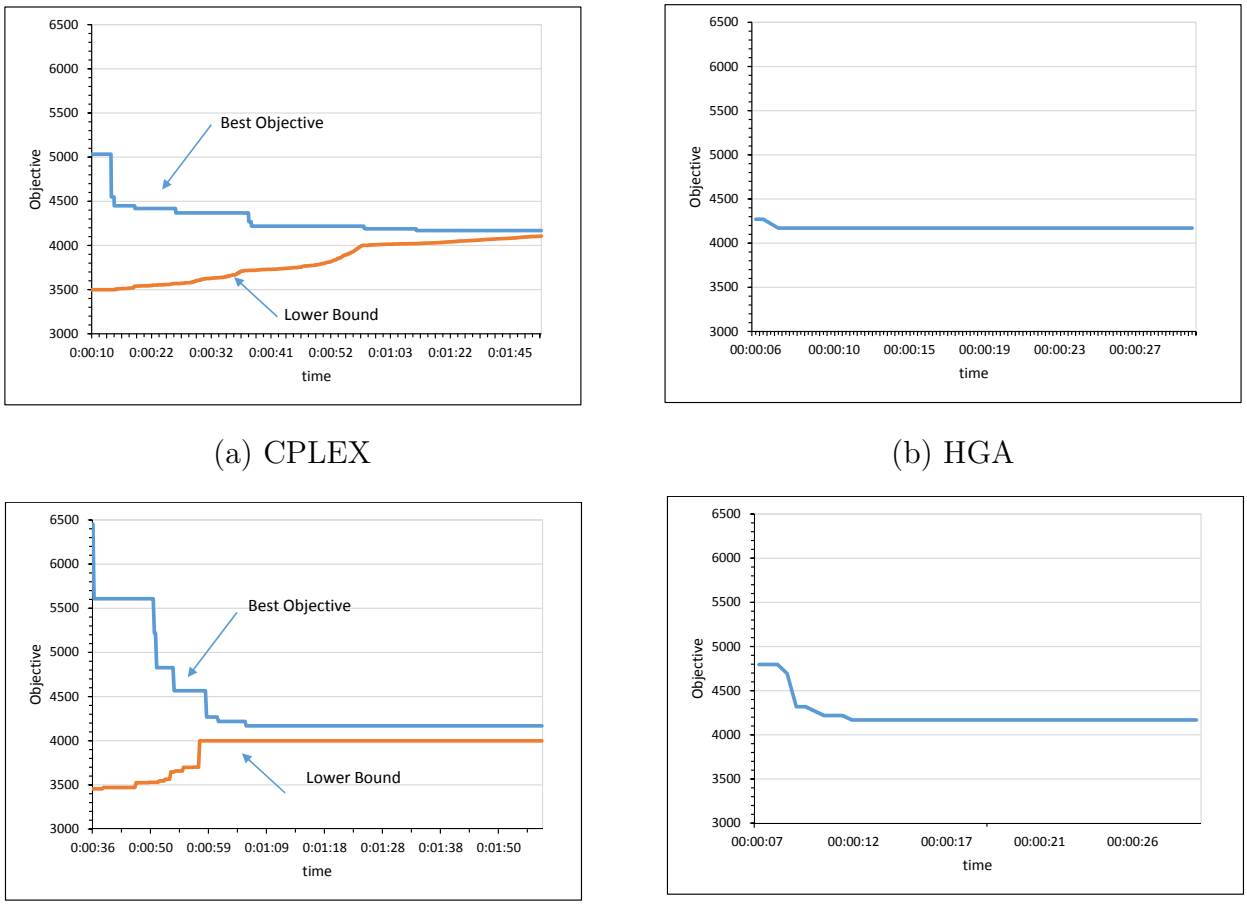
Figure 7: Precedence diagram for models A, B and C for Problem-1

Table 2: Operation time of each task in each model and task duplication cost in Problem-1

	Tasks			
Models	1	2	3	4
A	14	22	14	11
B	25	16	21	14
C	11	8	20	17
Task duplication cost	10	9	7	12

Figures 8-(a) and (b) show the convergence behavior of the CPLEX and that of the HGA, respectively, in solving the first version of Problem-1. CPLEX obtained the best objective

function value, 4169, in one minute and it declares this is an optimal solution at t= 0:02:23. In contrast, the HGA could reach the optimal value of 4169 only in just 7 seconds. For the second version of Problem-1 (see Figures 8-(c) and (d)), the best objective function value is 4169. CPLEX reached to this value in one minute. However, it was unable to declare this value to be an optimal value even after several hours of computation those this value is very close to the lower bound. The HGA reached the best objective function value of 4169 only in just 12 seconds. Hence, using this small size problem, this example demonstrates the correctness of the proposed HGA in solving the proposed mathematical model.



(a) CPLEX

(b) HGA

(c) CPLEX

(d) HGA

Figure 8: Convergence graph for CPLEX and HGA where (a) and (b) are for the first and (c) and (d) are for second versions of Problem-1

5.2.2. Problem-2

Problem-2 consists of the assembly of three models with sixteen tasks. The precedence diagrams for the three models in this problem are given in Figure 9. The tasks assembly times and their duplication costs are given in Table 3. The launching rate is set at six time-units. The maximum number of stations is equal to 10, and the station cost is equal to 500 cost units. The speed of the conveyor is equal to 1 unit distance per unit time. Similar to Problem-1, we

consider two versions of this problem. In the first version, the MPS has $d_A = 2$, $d_B = 1$ and $d_C = 2$ units. In this version of the problem, the total number of variables is 2695 from which 2585 them are integers. The total number of constraints is 5368. In the second version of this problem, the MPS has $d_A = 2$, $d_B = 3$ 1 and $d_C = 5$ units. This increases the number of variable to 5210 from which 5000 of them are integers. The total number of constraints also becomes 10633. This is nearly 100% increment in the size of the problem.

Figures 10-(a) and (b) provide the convergence of CPLEX and HGA, respectively, in solving the first version of Problem-2. The best objective function value obtained is 4250. CPLEX obtained this solution in three minutes whereas the HGA took only 40 seconds to arrive at this same solution. In solving the second version of Problem-2, where the size of the problem is increased by nearly 100%, CPLEX took about 1 hour and 40 minutes to arrive at the optimal solution (see Figure 10-(c)) with the objective value of 4359. This is more than 3000% of the time it requires to arrive at the best solution in the first version. This indicates that the computational time of CPLEX increases very rapidly as the problem size increases. However, as it can be seen in Figure 10 (d), this is not the case in HGA as it only requires less than 2 minutes to arrive at the optimal solution, just about double the amount of time it takes in the first version Problem-2. The convergence graph in Figure 10 (d) also illustrate the advantage of dividing the search into phases.

Table 3: Operation time of each task in each model and task duplication cost in Problem-2

	Tasks															
Models	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	2	2	0	3	0	3	0	0	2	0	0	0	0	3	0	2
B	1	0	2	0	2	0	4	3	0	2	3	3	5	3	4	3
C	1	3	0	0	4	3	0	1	3	4	0	4	0	2	0	1
Task duplication cost	10	11	14	12	9	15	10	9	9	10	8	13	14	11	20	10

5.2.3. Large problems

In this section, we consider additional two problems (Problems-3 and 4) which are much larger than the previous two problems. The purpose of this empirical example is to illustrate the fact that commercially available optimization packages cannot efficiently solve such large problems and to emphasize the need for developing a metaheuristic algorithm. The precedence diagrams for these large problems are given in Figures 11 and 12, respectively. The numbers of tasks in these problems are 48 and 72 where each problem has 4 models. The MPS was set as {1, 1, 1, 1} and {3, 1, 2, 1} in Problems-3 and 4, respectively. The maximum number of workstations was set to 28 in both problems. Other relevant data for these two problems are in given in Tables 4 and 5. The number of continues variable, binary variables, and constraints

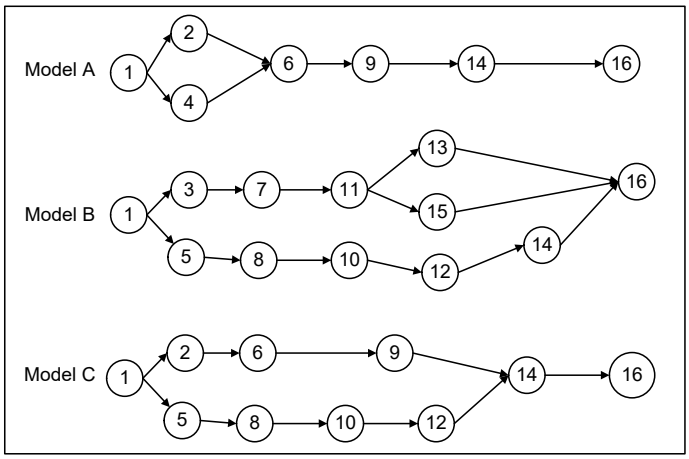
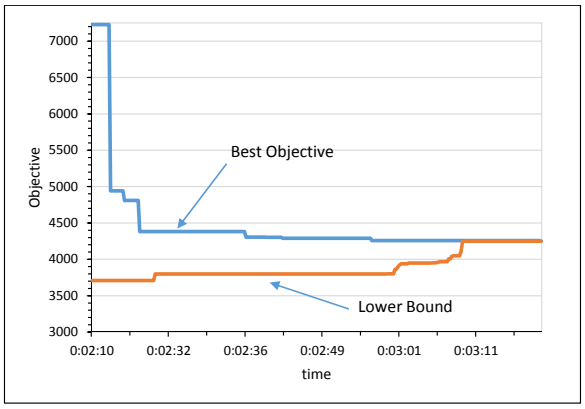
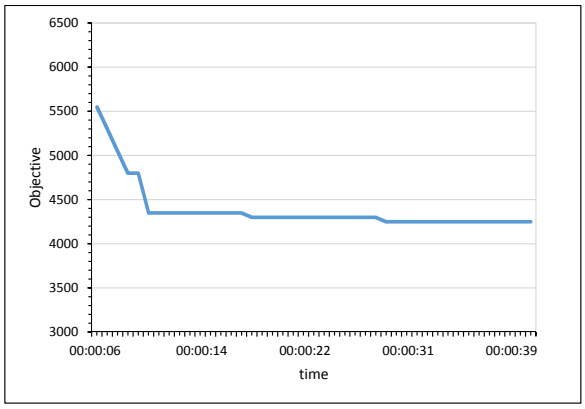


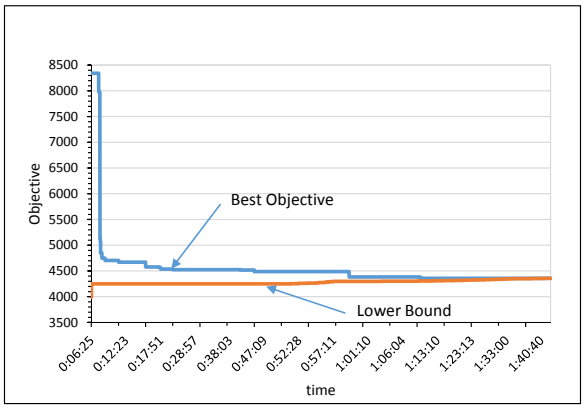
Figure 9: Precedence diagram for models A, B and C of Problem-2



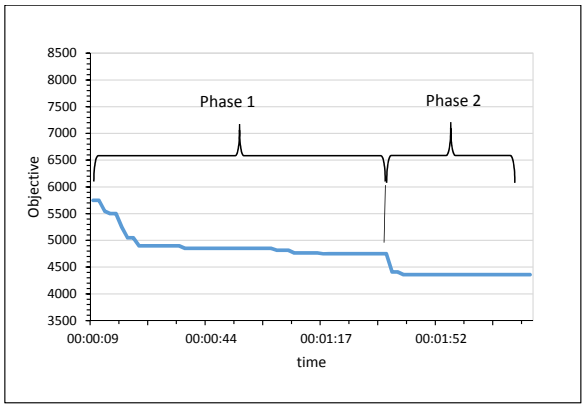
(a) CPLEX



(b) HGA



(c) CPLEX



(d) HGA

Figure 10: Convergence graph for CPLEX and HGA where (a) and (b) are for the first and (c) and (d) are for second versions of Problem-2

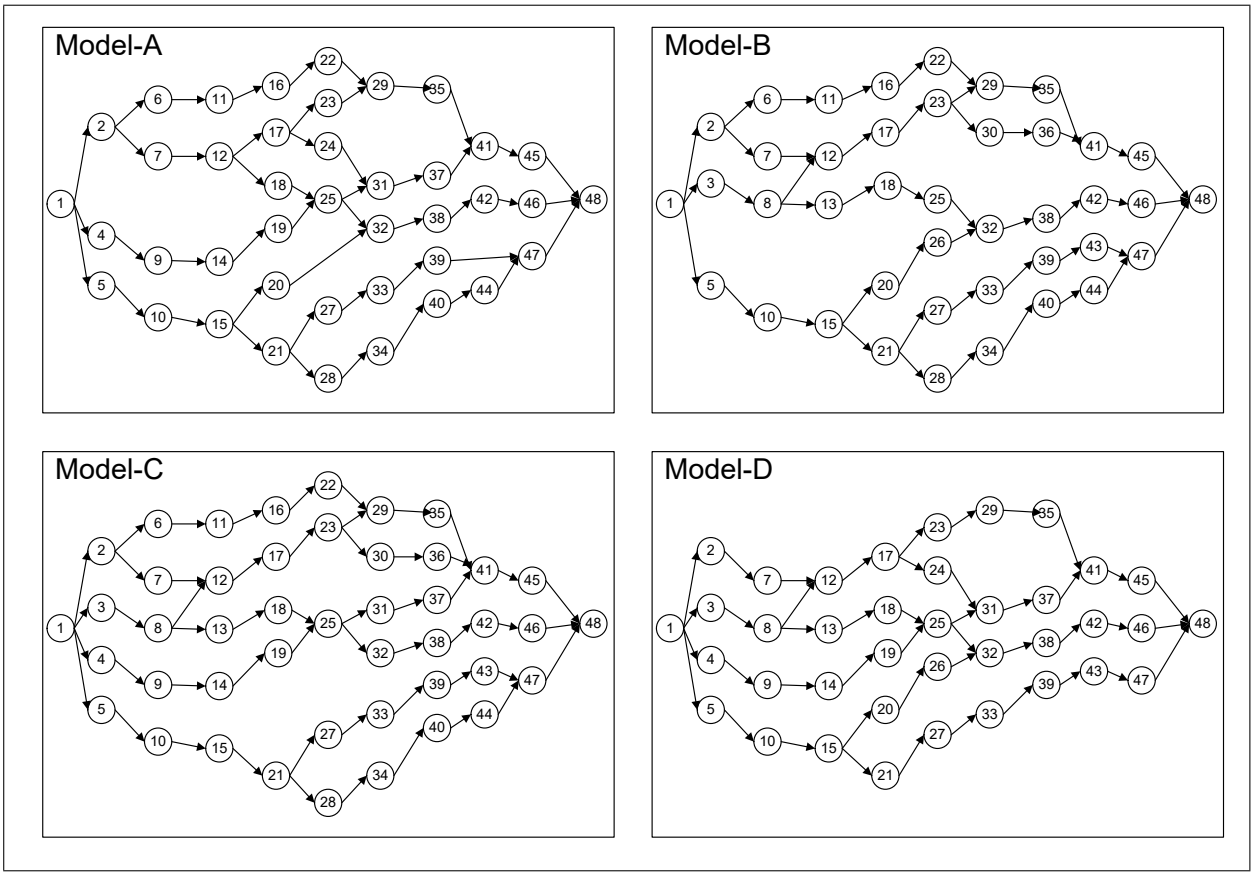


Figure 11: Precedence diagram for Problem-3

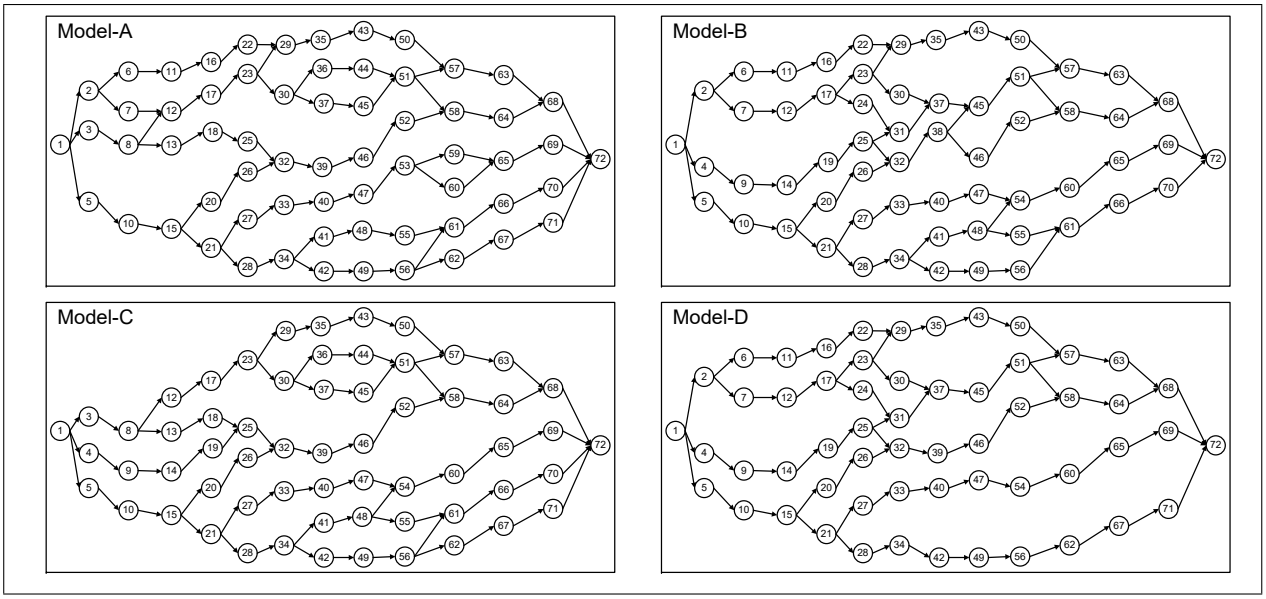


Figure 12: Precedence diagram for Problem-4

Table 4: Operation time of each task in each model and task duplication cost in Problem-3

	Tasks 1 to 24																							
Models	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
A	2	3	0	3	2	2	2	0	2	2	2	1	0	1	1	2	2	3	3	3	2	1	1	1
B	3	2	2	0	2	4	3	2	0	1	3	3	3	0	2	2	2	2	0	1	1	4	2	0
C	2	2	2	2	2	3	2	3	2	3	1	1	2	3	4	3	3	2	3	0	1	2	2	0
D	4	1	3	1	4	0	1	3	3	2	0	4	2	2	3	0	1	1	2	2	3	0	3	3
Task duplication cost	10	11	14	12	9	15	10	9	9	7	15	13	10	14	21	22	19	14	13	10	20	13	12	11

	Tasks 25 to 48																							
Models	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
A	2	0	2	2	5	0	1	2	3	4	3	0	2	1	3	3	2	2	0	2	2	3	4	5
B	1	1	3	2	2	2	0	3	3	2	5	2	0	1	1	1	2	2	2	2	3	3	2	2
C	2	0	2	2	1	2	1	4	1	3	3	1	2	2	2	4	2	5	3	4	2	2	2	3
D	2	3	2	0	3	0	4	2	1	0	2	0	3	3	4	0	3	3	2	0	1	1	3	4
Task duplication cost	7	8	9	15	12	15	16	18	20	19	20	12	13	14	15	16	10	12	7	8	10	11	14	16

Table 5: Operation time of each task in each model and task duplication cost in Problem-4

	Tasks 1 to 24																							
Models	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
A	2	8	7	0	6	6	5	6	0	7	8	8	5	0	5	5	6	6	0	6	8	8	6	0
B	2	5	0	8	8	5	6	0	8	7	5	5	0	7	6	5	8	0	5	5	6	5	4	7
C	2	0	5	6	4	0	0	6	8	8	0	8	4	6	8	0	4	4	7	7	8	0	4	0
D	2	7	0	6	7	4	8	0	6	7	7	8	0	6	5	5	7	0	5	7	8	8	7	6
Task duplication cost	13	20	18	18	16	20	19	19	13	17	10	14	10	13	16	15	11	18	17	14	18	20	17	10

	Tasks 25 to 48																							
Models	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
A	7	6	8	4	6	8	0	4	4	4	8	8	7	0	8	4	8	5	8	8	6	4	8	7
B	7	7	6	6	4	6	6	7	7	5	7	0	6	6	0	4	4	4	5	0	5	8	4	8
C	7	8	5	8	7	4	0	7	7	5	7	4	5	0	4	8	7	5	8	5	4	5	5	6
D	6	6	4	4	5	4	8	8	7	4	8	0	5	0	7	5	0	4	5	0	5	8	5	0
Task duplication cost	11	16	14	10	12	19	12	10	11	20	19	12	13	11	16	10	16	14	12	11	12	14	10	

	Tasks 49 to 72																							
Models	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72
A	6	4	6	6	7	0	7	5	4	4	8	6	7	8	8	4	6	4	7	8	4	8	6	6
B	4	7	6	8	0	8	6	8	8	7	0	8	8	0	4	6	7	8	0	5	4	5	0	6
C	8	8	4	4	0	7	4	8	4	8	0	5	8	7	7	6	5	6	5	5	5	6	6	6
D	5	5	6	8	0	5	0	4	4	6	0	5	0	6	4	8	8	0	7	8	8	0	6	6
Task duplication cost	15	17	13	20	16	16	12	18	12	13	12	20	14	15	18	14	20	12	14	20	15	12	16	20

are {252, 22892, 44940} and {420, 58520, 117541} in these two problems, respectively. Figure 13 provides the convergence graphs for both CPLEX and the proposed algorithm. As it can be seen from Figure 13-(a), CPLEX was able to generate the first incumbent solution after more than 6 hours of computation in solving Problem-3. The computation was continued for 36 hours and the best solution so far found has objective function value equal to 14330. On the other hand (see Figure 13-(c)), the proposed genetic algorithm was able to converge rapidly and found a slightly better solution in just a few minutes. For the larger problem (Problem 4), CPLEX was unable to obtain the first incumbent solution in more than 21 days of computation at which point we terminate the computation. For this same large problem, the GA was able to generate a feasible solution in a fraction of seconds, and it progressively improves this solution. To verify the correctness of the solution found by the GA, it was passed to CPLEX as starting incumbent and it was accepted as a feasible solution with the same objective function value as the one obtained by the GA. However, CPLEX was unable to further improve this solution after many hours of computation.

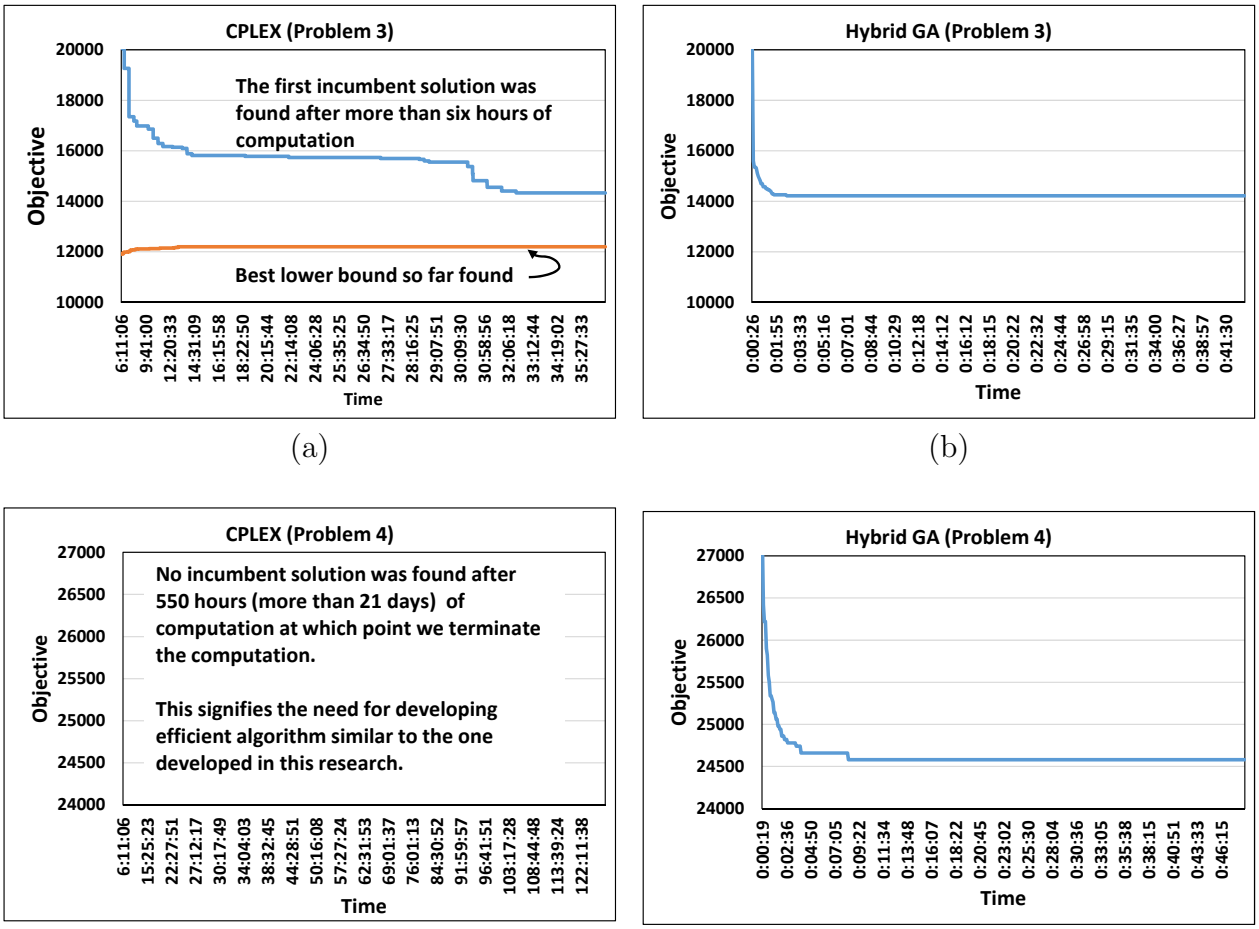


Figure 13: CPLEX vs hybrid GA in solving Problems-3 and 4

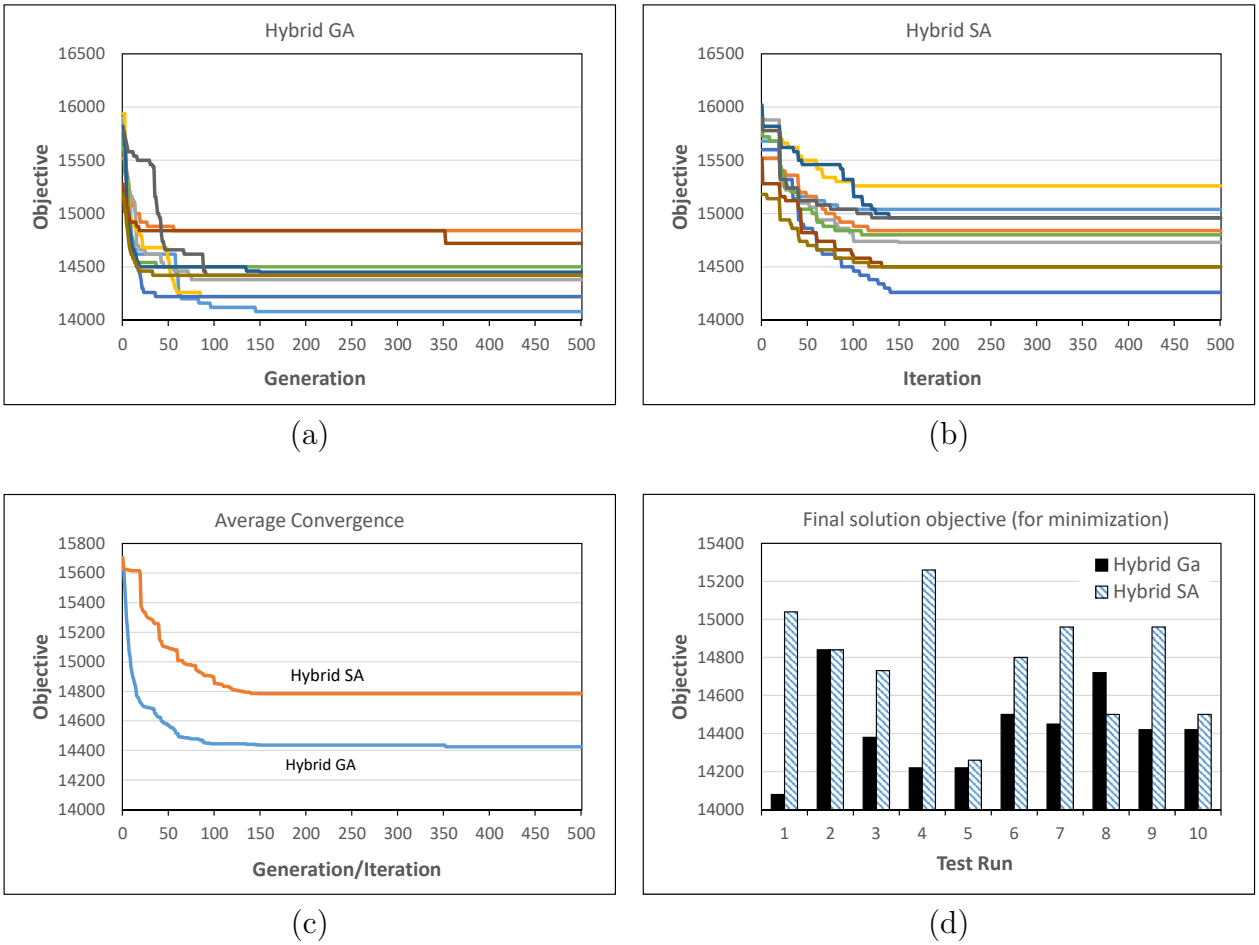
5.3. GA vs SA comparative study

It is generally required to compare the performance of a newly proposed algorithm against published results. However, this is not always possible, in particular, if the proposed algorithm is for a newly developed mathematical model. Solving a new mathematical model using a metaheuristic algorithm usually requires a new design of solution representation, initialization technique, implementation strategies, and search operators. To this end, we cannot directly compare the performance of the proposed genetic algorithm against published results. Instead, for the purpose of comparison, we develop a hybrid simulated annealing (hybrid-SA) algorithm. The SA utilizes the solution representation presented in Figure 2 and the linear programming subproblem in Section 2. The crossover operators are not applicable in the SA. However, the mutation operators of the hybrid-GA are used as the perturbation operators for the hybrid-SA. The overall structure of the developed SA is similar to the one presented in Defersha and Chen (2009) where the author attempted to solve an integrated cell formation and production planning problem using a multiple search path SA. In order to draw a balanced comparison, the number of search paths of the SA was set to be the same as the population size of the genetic algorithm. Each search direction follows a standard simulated annealing with the exception that the search paths communicate periodically for better results (see Defersha and Chen (2009)).

Figure 14 illustrates the convergence behaviors of the hybrid GA and hybrid SA in solving Problem-3. Using each algorithm, 10 test runs were conducted by varying the seed of the random number generator of the programming language used (C++ in Microsoft Visual Studio 2017). A simple inspection of Figures 14-(a) and (b) can easily reveal that GA has a faster and better convergence than the SA. This is more evident in Figure 14-(c) where the average convergence graphs from the ten test runs are plotted. The graphs show that GA convergence very fast and to a better solution. Figure 14-(d) shows the objective function of the final solutions from each algorithm in ten test runs. As it can be seen from this figure, except for run 8, GA outperforms SA in all the other runs. Comparable results were also obtained in solving Problem-4 as it is depicted in Figure 15.

5.4. Design of Experiment - Algorithm Robustness

In this section, we investigate the effects of the cross-over and mutation rates (probabilities) on the performance of the proposed algorithm. We initially perform many preliminary test runs. From the preliminary runs, we were able to determine that the genetic algorithm works better for higher values of cross-over probabilities (roughly 0.7 to 0.98) and lower mutation rates (roughly from 0.05 to 0.2). This is well in agreement with many published results in using a genetic algorithm. Once we identify preferable ranges for the cross-over and mutation rates, we perform Analysis of Variance (ANOVA) to investigate the robustness of the algorithm to its parameter settings. The ANOVA was conducted on serval problems. In here, we will discuss



(a) (b)

(c) (d)

Figure 14: Hybrid GA vs hybrid SA in Solving Large Problem-3

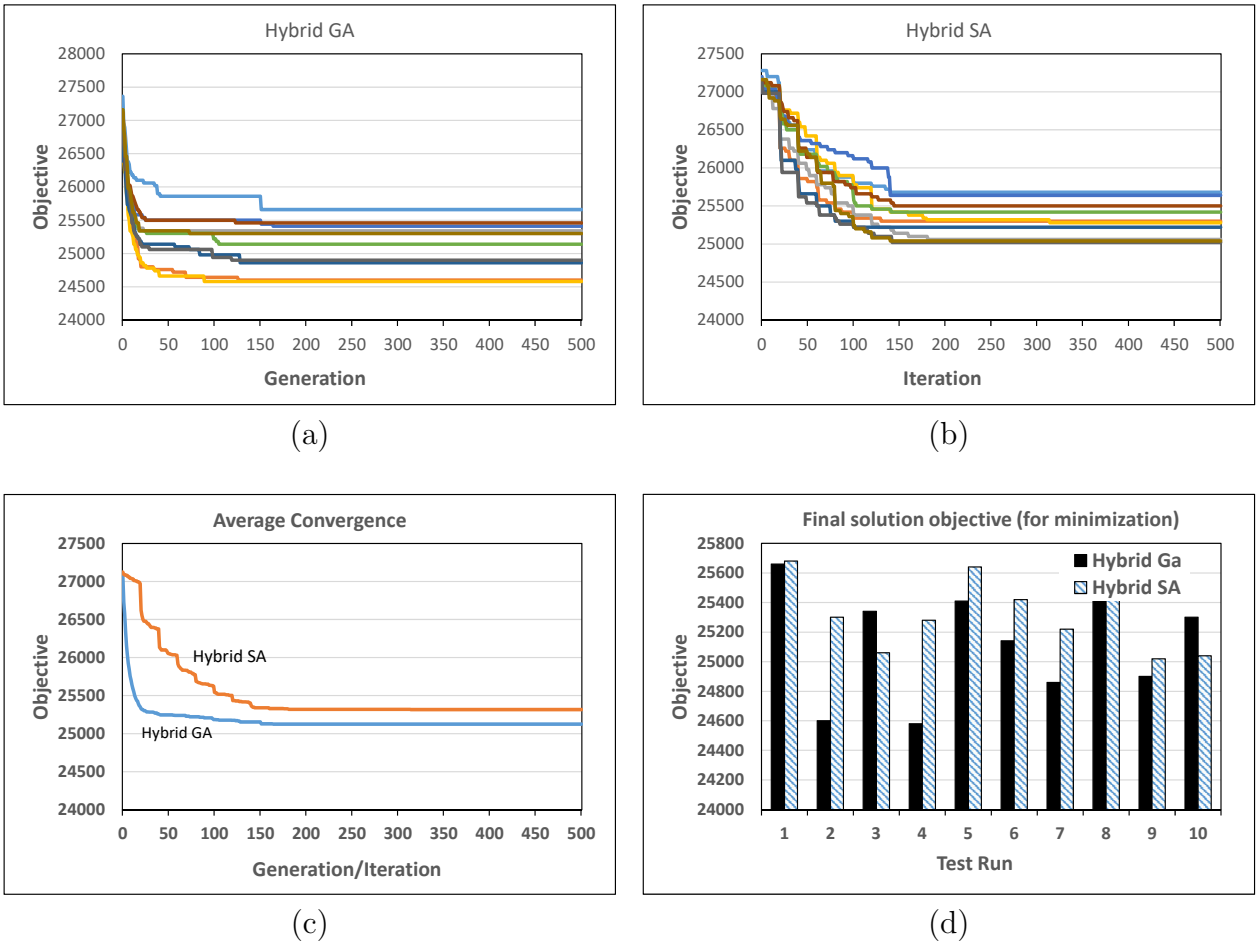


Figure 15: Hybrid GA vs hybrid SA in Solving Large Problem-3

the analysis in solving only Problem-3 as the other results are very similar. The data for the ANOVA is given in Table 6. The cross-over and mutations rates were set at three leaves as {0.75, 0.85, 0.95} and {0.10, 0.15, 0.20}, respectively. For each parameter combination, the problem was solved four times by varying the seed of the random number generator of the programming environment. This gives a total of four replicates for each test run.

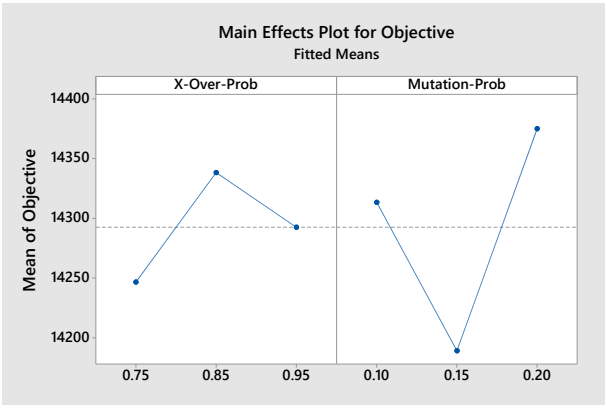
The main effect plot in Figure 16-(a) shows neither positive nor negative trend while the factors (the probabilities) are changing. This indicates that, within the identified ranges, the change of factors has no significant effect on the performance of the algorithm. This is also indicated by the p-values of 0.776 and 0.349 (see Figure (d)) which are much higher than the significance level $\alpha = 0.05$ used in the experiment. As it can also be seen in Figure 16-(b), except for one point, the interaction plots are nearly parallel to each other. This indicates that there is no significant interaction of the factors. The corresponding p-value for the interaction effect is 0.95. The contour plot in 16-(c) shows a wider combination of parameters in which the algorithm works with similar performance. The above result is a clear evidence that the algorithm is very robust to the parameters. The normal probability and the residual plots (Figure 16-c and d, respectively) indicate the adequacy of the test as they are not showing unusual patterns.

Table 6: ANOVA data for analysing the effect of cross-over and mutation probabilities in solving Problem-3

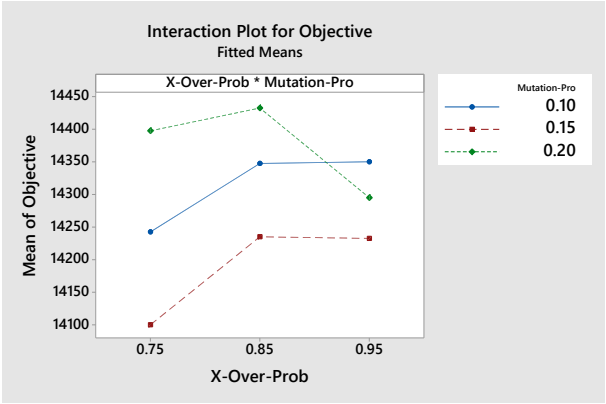
Experimental Run	Cross-Over Probability	Mutation Probability	Objective function value for Replications			
			1	2	3	4
1	0.75	0.10	13840	14380	14340	14410
2	0.75	0.15	13720	14260	14180	14240
3	0.75	0.20	14800	14420	14030	14340
4	0.85	0.10	13800	14540	14760	14290
5	0.85	0.15	14080	14260	14800	13800
6	0.85	0.20	14180	14380	14840	14330
7	0.95	0.10	14220	14340	14840	14000
8	0.95	0.15	14340	14420	13800	14370
9	0.95	0.20	14260	14420	14250	14250

6. Discussion and Conclusions

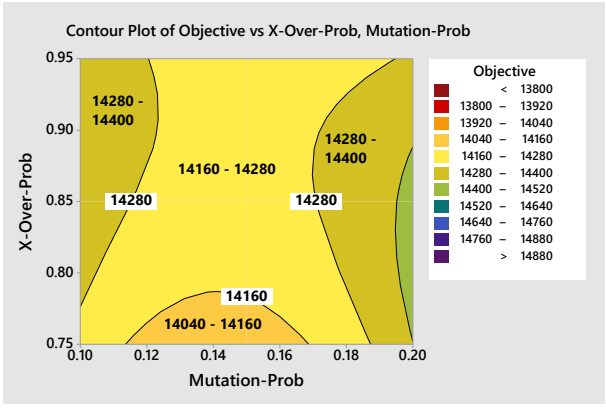
In this paper, we developed a mixed integer linear programming model to simultaneously solve balancing and sequencing problems in the mixed-model assembly line on a continuously moving conveyor. The main objectives of the proposed model are minimizing the length and number of workstations, minimizing the workstations cost, and minimizing the tasks duplication cost. The task duplication cost arises from the fact that common tasks between various models can be assigned to different workstations with task duplication costs. The proposed



(a)



(b)

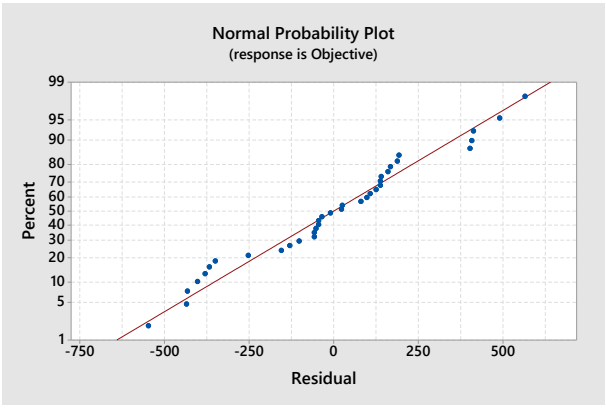


(c)

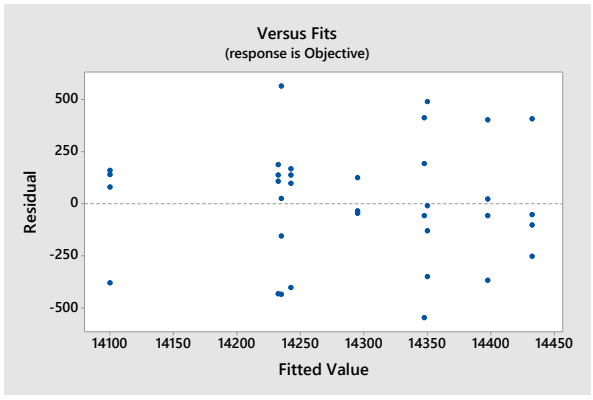
ANOVA Output

Source	DF	Adj SS	Adj MS	F-Value	P-Value
X-Over-Prob	2	50417	25208	0.26	0.776
Mutation-Prob	2	215017	107508	1.09	0.349
X-Over-Prob*Mutation-Prob	4	68267	17067	0.17	0.950
Error	27	2654975	98332		
Total	35	2988675			

(d)



(e)



(f)

Figure 16: Results of ANOVA on the effect of cross-over and mutation probabilities in solving Problem-3

model is NP-hard. Hence, to overcome the computational difficulties in solving large size problems, we developed an efficient linear programming embedded hybrid genetic algorithm. In the genetic algorithm, a solution representation encodes the model sequence as well as task-assignment to different workstations. Therefore, corresponding to a particular solution visited by the GA, the binary variables were obtained by decoding the solution representation. The continuous variables corresponding to the integer solution were determined by solving a linear programming subproblem. Numerical examples were presented to illustrate the model and the computational efficiency of the developed hybrid genetic algorithm. The performance of the developed GA was compared with the branch and bound algorithm. Based on the obtained results, our proposed HGA outperformed the BB algorithm in finding an acceptable good solution, which was near to the optimal solution, in much less computational time. The comparative study between GA and SA reveals that the former is a preferred approach in solving the proposed model. The proposed algorithm is also robust to its main parameter settings as it is indicated by the ANOVA. Our future work in this area includes changing the type of operation time from deterministic, which has been considered in this study, to stochastic, exploiting parallel workstations and zoning constraints in the model.

References

Akpinar, S. and Baykasoglu, A. (2014). Modeling and solving mixed-model assembly line balancing problem with setups. part i: A mixed integer linear programming model. *Journal of manufacturing systems*, 33(1):177–187.

Bautista, J., Alfaro-Pozo, R., and Batalla-García, C. (2016). Grasp for sequencing mixed models in an assembly line with work overload, useless time and production regularity. *Progress in Artificial Intelligence*, 5(1):27–33.

Bautista, J., Cano, A., and Alfaro-Pozo, R. (2017). A hybrid dynamic programming for solving a mixed-model sequencing problem with production mix restriction and free interruptions. *Progress in Artificial Intelligence*, 6(1):27–39.

Becker, C. and Scholl, A. (2006). A survey on problems and methods in generalized assembly line balancing. *European journal of operational research*, 168(3):694–715.

Bock, S., Rosenberg, O., and van Brackel, T. (2006). Controlling mixed-model assembly lines in real-time by using distributed systems. *European Journal of Operational Research*, 168(3):880–904.

Boysen, N., Fliedner, M., and Scholl, A. (2007). A classification of assembly line balancing problems. *European journal of operational research*, 183(2):674–693.

Boysen, N., Fliedner, M., and Scholl, A. (2009). Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, 192(2):349–373.

Bukchin, Y. and Rabinowitch, I. (2006). A branch-and-bound based solution approach for the mixed-model assembly line-balancing problem for minimizing stations and task duplication costs. *European Journal of Operational Research*, 174(1):492–508.

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

Faccio, M., Gamberi, M., and Bortolini, M. (2016). Hierarchical approach for paced mixed-model assembly line balancing and sequencing with jolly operators. *International Journal of Production Research*, 54(3):761–777.

Guo, G. and Ryan, S. M. (2017). Risk-averse stochastic integer programs for mixed-model assembly line sequencing problems.

Ho, J. (2005). A proposed approach for reconfiguration of flexible assembly line systems by motion genes. *International journal of production research*, 43(9):1729–1749.

Hwang, R. and Katayama, H. (2010). Integrated procedure of balancing and sequencing for mixed-model assembly lines: a multi-objective evolutionary approach. *International Journal of Production Research*, 48(21):6417–6441.

Hyun, C. J., Kim, Y., and Kim, Y. K. (1998). A genetic algorithm for multiple objective sequencing problems in mixed model assembly lines. *Computers & Operations Research*, 25(7):675–690.

Ishigaki, A. and Miyashita, T. (2016). Development of search method for sequencing problem in mixed-model assembly lines. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 10(3):JAMDSM0048–JAMDSM0048.

Kim, Y. K., Kim, J. Y., and Kim, Y. (2000). A coevolutionary algorithm for balancing and sequencing in mixed model assembly lines. *Applied intelligence*, 13(3):247–258.

Kucukkoc, I. and Zhang, D. Z. (2016a). Integrating ant colony and genetic algorithms in the balancing and scheduling of complex assembly lines. *The International Journal of Advanced Manufacturing Technology*, 82(1-4):265–285.

Kucukkoc, I. and Zhang, D. Z. (2016b). Mixed-model parallel two-sided assembly line balancing problem: A flexible agent-based ant colony optimization approach. *Computers & Industrial Engineering*, 97:58–72.

Makarouni, I., Siskos, E., and Psarras, J. (2016). Multiobjective large scale job sequencing optimization based on a synergy of compensatory and non compensatory methods. *Operational Research*, 16(2):223–244.

Manavizadeh, N., Rabbani, M., Moshtaghi, D., and Jolai, F. (2012). Mixed-model assembly line balancing in the make-to-order and stochastic environment using multi-objective evolutionary algorithms. *Expert Systems with Applications*, 39(15):12026–12031.

Manavizadeh, N., Rabbani, M., and Radmehr, F. (2015). A new multi-objective approach in order to balancing and sequencing u-shaped mixed model assembly line problem: a proposed heuristic algorithm. *International Journal of Advanced Manufacturing Technology*, 79.

McMullen, P. R. and Frazier, G. V. (2000). A simulated annealing approach to mixed-model sequencing with multiple objectives on a just-in-time line. *Iie Transactions*, 32(8):679–686.

Monteiro Filho, B., Albuquerque, I., and Neto, F. L. (2017). Simultaneously solving mixed model assembly line balancing and sequencing problems with fss algorithm. *arXiv preprint arXiv:1707.06185*.

Mosadegh, H., Ghomi, S. F., and Zandieh, M. (2012a). Simultaneous solving of balancing and sequencing problems in mixed-model assembly line systems. *International Journal of Production Research*, 50(18):4994–5016.

Mosadegh, H., Zandieh, M., and Ghomi, S. F. (2012b). Simultaneous solving of balancing and sequencing problems with station-dependent assembly times for mixed-model assembly lines. *Applied Soft Computing*, 12(4):1359–1370.

Rabbani, M., Montazeri, M., Farrokhi-Asl, H., and Rafiei, H. (2016a). A multi-objective genetic algorithm for a mixed-model assembly u-line balancing type-i problem considering human-related issues, training, and learning. *Journal of Industrial Engineering International*, 12(4):485–497.

Rabbani, M., Siadatian, R., Farrokhi-Asl, H., and Manavizadeh, N. (2016b). Multi-objective optimization algorithms for mixed model assembly line balancing problem with parallel workstations. *Cogent Engineering*, 3(1):1158903.

Roshani, A., Roshani, A., Ghazi Nezami, F., and Ghazi Nezami, F. (2017). Mixed-model multi-manned assembly line balancing problem: a mathematical model and a simulated annealing approach. *Assembly Automation*, 37(1):34–50.

Saif, U., Guan, Z., Liu, W., Wang, B., and Zhang, C. (2014). Multi-objective artificial bee colony algorithm for simultaneous sequencing and balancing of mixed model assembly line. *The International Journal of Advanced Manufacturing Technology*, 75(9-12):1809–1827.

Sawik, T. (2002). Monolithic vs. hierarchical balancing and scheduling of a flexible assembly line. *European Journal of Operational Research*, 143(1):115–124.

Scholl, A., Klein, R., and Domschke, W. (1998). Pattern based vocabulary building for effectively sequencing mixed-model assembly lines. *Journal of Heuristics*, 4(4):359–381.

Simaria, A. S. and Vilarinho, P. M. (2004). A genetic algorithm based approach to the mixed-model assembly line balancing problem of type ii. *Computers & Industrial Engineering*, 47(4):391–407.

Vilarinho, P. M. and Simaria, A. S. (2006). Antbal: An ant colony optimization algorithm for balancing mixed-model assembly lines with parallel workstations. *International journal of production research*, 44(2):291–303.

Yagmahan, B. (2011). Mixed-model assembly line balancing using a multi-objective ant colony optimization approach. *Expert Systems with Applications*, 38(10):12453–12461.