We developed mixed integer programming (MIP) models and hybrid genetic-local search algorithms for the scheduling problem of unrelated parallel machines with job sequence and machine-dependent setup times and with job splitting property. The first contribution of this paper is to introduce novel algorithms which make splitting and scheduling simultaneously with variable number of subjobs. We proposed simple chromosome structure which is constituted by random key numbers in hybrid genetic-local search algorithm (GAspLA). Random key numbers are used frequently in genetic algorithms, but it creates additional difficulty when hybrid factors in local search are implemented. We developed algorithms that satisfy the adaptation of results of local search into the genetic algorithms with minimum relocation operation of genes’ random key numbers. This is the second contribution of the paper. The third contribution of this paper is three developed new MIP models which are making splitting and scheduling simultaneously. The fourth contribution of this paper is implementation of the GAspLAMIP. This implementation let us verify the optimality of GAspLA for the studied combinations. The proposed methods are tested on a set of problems taken from the literature and the results validate the effectiveness of the proposed algorithms.

1. Introduction

This study focuses on the problem of scheduling jobs that involve splitting and machine- and sequence-dependent setup times on nonidentical (unrelated) parallel machines to minimize the maximum completion time (makespan). The problem will be referred to as $R_m / S_{ijk} / C_{\text{max}}$ with job splitting. Job splitting is a great requirement for some industries. For example, drilling of printed circuit board (PCB) manufacturing, dicing of semiconductor wafer manufacturing, and weaving of textile manufacturing are major bottleneck operations and splitting the jobs into subjobs and processing them on different machines are necessary because of the due date pressure and competitive challenges of the modern manufacturing industry. In fact, the methods that were proposed in this paper can be applied to real weaving process of textile manufacturing industry from which this research is originated.

The mentioned problem has a set of $n$ jobs ($N = \{1, \ldots, n\}$), which can be split into subjobs. The number of subjobs for each job will be between one and the maximum number of subjobs. The maximum number of subjobs will be defined beforehand and the proposed algorithms will try to find the best number of subjobs for each job. More setup times would be required if there are more subjobs. Hence, in the scheduling problem considered in this study, finding an appropriate number of subjobs to be split from each job is so important.

If the jobs’ processing times are dependent on the assigned machines and there is no relationship among these machines, then the machines are considered to be unrelated [1]. So, the processing time of Job$_j$ is different from Job$_k$ on Machine$_i$ in unrelated machine environment. As it will be explained in next sections, $p_{ij}$ indicates processing time of Job$_j$, at Machine$_i$. The mentioned subjobs must be processed on one machine in a set $M = \{1, \ldots, m\}$ of $m$ unrelated parallel machines ($R_m$). The setup times are sequence- and machine-dependent ($S_{ijk}$). Each machine has its own matrix of setup times, and these matrices are different from each other. The setup time on Machine$_i$ between Job$_j$ and Job$_k$ is
different compared with the setup time between Job$_k$ and Job$_j$ on the same machine.

In scheduling theory, the makespan (\(C_{\text{max}}\)) is defined as the completion time of the final job (when it leaves the system). A smaller \(C_{\text{max}}\) implies a higher utilization, which is closely related to the throughput rate of the system. Therefore, reducing \(C_{\text{max}}\) will lead to a higher throughput rate [2]. For that reason, minimization of the “makespan” is the objective of this study.

Minimizing the makespan on a scheduling problem with \(m\) identical parallel machines and sequence-dependent setup times is NP-hard [3]. Thus, a more complex case of the problem with job splitting and unrelated parallel machines is also NP-hard.

The present paper contains mixed integer programming (MIP) models and hybrid genetic-local search algorithm techniques to solve the mentioned \(R_m/S_{ijk}/C_{\text{max}}\) with job splitting problem. The proposed algorithms in this study perform job splitting and scheduling simultaneously with variable number of subjobs, where, to the best of our knowledge, no work has been published on an algorithm with these properties. This is the first contribution of this paper. The second contribution of this work is getting over the problem of main difficulty in using random key numbers in the chromosome for hybrid structures. We proposed simple chromosome structure which is constituted by random key numbers in hybrid genetic-local search algorithm (GAspLA). The literature random key numbers are frequently used in chromosome. But local search application inside the genetic algorithm made the situation more complex. Inside each generation of the genetic algorithm, we are trying to find better job sequence using local search. But the better job sequence must be adapted to the chromosome which will be used in each next generation. We need to determine which data in genes will be exchanged or in which condition and in which genes we will regenerate new random key numbers. To manage the random key numbers according to the desired job sequence that would be necessary during local search operations, which is implemented in the genetic algorithm, we developed algorithms that satisfy the adaptation of results of local search into the genetic algorithms with minimum relocation operation of genes' random key numbers. The third contribution of this paper is three developed mixed integer programming (MIP) models which are designed to solve different subjob environments. In the MIP1 and the MIP2, numbers of the subjobs will be determined by models between one and the maximum number of subjob. In the MIP3, the numbers of subjobs will be determined randomly between one and maximum number of subjobs beforehand and will be given as an input data to the model. In the MIP1, the quantities of subjobs are not equal. In the MIP2 and MIP3 the quantities of subjobs are equal. Finally we presented an implementation (GAspLAMP) which is the fourth contribution of this study. In this implementation, obtained scheduling using GAspLA is combined with the MIP formulation. So, the result of the GAspLA feeds MIP formulation with initial solution set. This implementation lets us verify the optimality of GAspLA for the studied combinations. To present the performance of the algorithms, the problem set SchedulingResearch 2005 [4] was used. The results demonstrated that proposed algorithms outperformed the compared ones.

This paper is organized as follows. Section 2 provides a review of the existing literature. In Section 3, mixed integer programming (MIP) mathematical models are formulated. In Section 4, the proposed GAspLA is described. In Section 5, the implementation of GAspLAMIP is presented. In Section 6, the experimental design and computational results are reported. Section 7 concludes the paper.

2. Literature Review

The interest in scheduling problems with setup times began in the mid-1960s. These types of problems have received continuous interest from researchers since then. Allahverdi et al. [5] presented a survey of scheduling problems with setup times or costs. The paper classified scheduling problems into those with batching and nonbatching considerations and those with sequence-independent and sequence-dependent setup times. It also classified the literature according to shop environments, including a single machine, parallel machines, flow shop, no-wait flow shop, flexible flow shop, job shop, and open shop. Machine setup time is a significant factor for production scheduling in manufacturing environments, and sequence-dependent setup times have been investigated by researchers. Zhu and Wilhelm [6] presented a review of scheduling problems that involves sequence-dependent setup times (costs). Li and Yang [7] presented a review on nonidentical parallel-machine scheduling research in which the total weighted completion times are minimized. Models and relaxations are classified in this paper, and heuristics and optimizing techniques are surveyed for the problems. Lin et al. [8] studied research that compares the performance of various heuristics for unrelated parallel machine scheduling problems. They proposed a metaheuristic, and computational results showed that the proposed metaheuristic outperformed other existing heuristics for each of the three objectives, minimized makespan, total weighted completion time, and total weighted tardiness, when run with a parameter setting that is appropriate for the objective.

Because of the problem complexity, it is general practice to find an appropriate heuristic rather than an optimal solution for the parallel-machine scheduling problem. Park et al. [9] used a neural network and heuristic rules to schedule jobs with sequence-dependent setup times on parallel machines. To calculate the priority index of each job, they utilized a neural network, and their computational results showed that the proposed approach outperformed the Lee et al. [10] original ATCS (Apparent Tardiness Cost with Setups) and a simple application of ATCS. van Hop and Nagarur [11] focused on scheduling problems of \(n\) printed circuit boards (PCBs) for \(m\) nonidentical parallel machines, and a composite genetic algorithm was developed to solve this multiobjective problem. Test results of the proposed methodology showed that the solutions were efficient and were obtained within a reasonable amount of time. Zandieh et al. [12] studied on the hybrid flow shop scheduling problems.
in which there were sequence-dependent setup times. They proposed an immune evolutionary algorithm (IEA) for this problem. In the paper of Wang [13] the problem of single machine common due date scheduling with controllable processing times is considered. Behnamian et al. [14] compared makespan results solved by ant colony optimization (ACO), variable neighborhood search (VNS), simulated annealing (SA), and the VNS hybrid algorithm for the problem of parallel machine scheduling problems with sequence-dependent setup times. Yang [15] proposed an evolutionary simulation optimization approach for solving the parallel-machine scheduling problem. The proposed methodology’s findings were benchmarked against lower-bound solutions, and efficient results were obtained. Balin [16] attempted to findings were benchmarked against lower-bound solutions, and efficient results were obtained. Balin [16] attempted to adapt a GA to the nonidentical parallel machine scheduling problem and proposed an algorithm with a new crossover operator and a new optimality criterion. The new algorithm was tested on a numerical example by implementing it in simulation software. The results showed that, in addition to its high computational speed for a larger-scale problem, the GA addressed the nonidentical parallel machine scheduling problem of minimizing the makespan. Keskinurk et al. [17] aimed to minimize the average relative percentage of imbalance and used the ACO algorithm for load balancing in parallel machines with sequence-dependent setup times. The results of tests on various random data showed that the ACO outperformed the GA and heuristics.

In this section, some of the previous studies on the unrelated parallel machine scheduling problem with the objective of minimizing the makespan are discussed. All of these studies considered in the following assumptions:

(i) machine-dependent and job sequence-dependent setup times;
(ii) all of the jobs are available at time zero.

A metaheuristic Meta-RaPS was introduced by Rabadi et al. [18], and its performance was evaluated by comparing its solutions to those obtained by existing heuristics. The results of the metaheuristic showed that the solutions were efficient. A two-stage ant colony optimization (ACO) algorithm was proposed by Arnaout et al. [19]. The performance of this metaheuristic was evaluated by using the benchmark problems, and the solutions were found to be efficient. Another method was suggested by Chang and Chen [20] for the same NP-hard problem. A set of dominance properties was developed, including intermachine and intramachine switching properties, which are necessary conditions of job sequencing orders in a derived optimal schedule. They also introduced a new metaheuristic by integrating the dominance properties with a genetic algorithm (GADP). The performance of this metaheuristic was evaluated by using benchmark problems from the literature, and the solutions were efficient. Vallada and Ruiz [1] proposed a genetic algorithm that includes the crossover operator with a limited local search as well as a fast local search procedure; this method was tested on both small and large problem sets and outperformed the other evaluated methods. Scheduling Research 2005 [4] datasets were used by many of the researchers [18–20] in the literature. Yilmaz Eroglu et al. [21] proposed a genetic algorithm with local search (GALA) that was based on random keys. To present the performance of the algorithm, the same problem set Scheduling Research 2005 [4] was used. The results showed that the GALA, which is the foundation of this study, outperformed the other algorithms.

The jobs that are considered in this paper can be split into subjobs; a feature that is very seldom studied in the literature. Studies can be categorized as splitting the job into subjobs of discrete units or continuous units. Some studies that involve discrete units are as follows: Kim et al. [22] focused on the dicing of semiconductor wafer manufacturing, which is the major bottleneck operation of the whole process. They proposed a simulated annealing algorithm for the problem of allotting work parts of L jobs into M parallel unrelated machines, where a job is referred to a lot composed of N items. Setup times were job sequence-dependent. The proposed SA method outperformed a neighborhood search method in terms of the total tardiness. Kim et al. [23] suggested a two-phase heuristic algorithm for the problem of scheduling a drilling process in the PCB manufacturing system. It was assumed that a job can be split into a discrete number of subjobs and that they would be processed on identical parallel machines independently. In the first phase of the algorithm, an initial sequence is generated by existing heuristic methods for the parallel machine scheduling problem. In the second phase, each job was split into subjobs, and then jobs and subjobs were rescheduled on the machines by using a certain method. The performance of the suggested algorithm was proved by the results of the computational experiments, which performed better than an existing method. Shim and Kim [24] also focused on PCB manufacturing system bottleneck operations of the drilling process. For the problem of scheduling jobs that can be split into subjobs, they developed several dominance properties and lower bounds and then suggested a branch and bound algorithm using them. The suggested algorithm solved problems of moderate size in a reasonable amount of computational time. Xing and Zhang [25] proposed a heuristic algorithm for the P/split/Cmax problem and analyzed the worst case performance of the algorithm. Yalaoui and Chu [3] considered a simplified real-life identical parallel machine scheduling problem with sequence-dependent setup times and job splitting to minimize the makespan. The proposed method composed of two phases. In the first phase, the problem was reduced to a single machine scheduling problem and was transformed into a traveling salesman problem (TSP), which could efficiently be solved by using Little’s method. In the second phase, a derived initial solution was improved in a step-by-step manner, accounting for the setup times and job splitting. Tahar et al. [26] proposed a new method, which is an improvement of the method proposed by Yalaoui and Chu [3]. For the problem of splitting the job into continuous units, Serafini [27] studied the scheduling problem of looms in the textile industry. Jobs might be independently split over several specified machines, and preemption was allowed. It was assumed that there were uniform parallel machines and that there were no setup times. For the mentioned problem, heuristic algorithms were
proposed for the objective of minimizing the maximum weighted tardiness. Yilmaz Eroglu et al. [28] proposed a genetic algorithm (without local search) that decided number of subjobs for each order and makes schedule simultaneously. Result of computations showed that the suggested algorithm could find solutions for problems with 75 machines and 111 jobs in a reasonable amount of CPU time. According to the results, the proposed GA outperformed existing system in makespan. Pimenteletal.[29] focused on the problem which is related with the knitting production process. MIP model was formulated and the heuristics and local search algorithms were proposed for the identical parallel machine scheduling problem with job splitting to minimize total tardiness.

3. MIP Mathematical Models

In this section, mixed integer programming (MIP) mathematical models are formulated to find optimal solutions for the unrelated parallel machine scheduling problem with sequence dependent setup times and job splitting. Three mathematical models are developed on this topic. In the MIP1 and the MIP2, numbers of the subjobs will be determined by models between one and the maximum number of subjob. In the MIP3, the numbers of subjobs will be determined randomly between one and maximum number of subjobs beforehand and will be given as an input data to the model. In the MIP1, the quantities of subjobs are not equal. In the MIP2 and MIP3 the quantities of subjobs are equal.

Mentioned problem has a set of \( n \) jobs \( (N = \{1, \ldots, n\}) \) which can be split into subjobs. The numbers that are declared in a set of \( B = \{B[1], \ldots, B[n]\} \) have different meaning in MIP1, MIP2, and MIP3.

The numbers in set \( B \) for MIP1 and MIP2 denote the maximum number of subjobs for each job. MIP1 and MIP2 will decide the number of subjobs between one and maximum number of subjobs which is defined in set of \( B \). So, in MIP1 and MIP2 the numbers of subjobs are variable.

The numbers in set \( B \) for MIP3 will be predefined randomly between one and the maximum number of subjobs for each job. So, in MIP3 the numbers of subjobs are constant. But this constant number will be generated randomly between one and the maximum number of subjobs in order to approximate the hybrid genetic-local search algorithm which will be described in Section 4.

Mentioned subjobs have to be processed on one machine in a set \( M = \{1, \ldots, m\} \) of \( m \) unrelated parallel machines \( (R_m) \):

(i) \( p_{ij} \): processing time of Job \( j \), \( j \in N \) at Machine \( i \), \( i \in M \).
(ii) \( S_{ijk} \): machine based sequence dependent setup time on Machine \( i \), \( i \in M \), when processing Job \( k \), \( k \in N \), after having processed Job \( j \), \( j \in N \).

Details about the models are explained in the following subsections.

3.1. MIP1: The Quantities of Subjobs Are Not Equal. The quantities of subjobs are not equal in MIP1. This is the point of difference of this model from hybrid genetic-local search algorithm (GAspLA) that will be explained in Section 4:

(i) \( q^p \): total constant production quantity for each job.
(ii) \( D_{ij} = p_{ij}/q^p \): processing time of unit quantity of Job \( j \) at Machine \( i \), \( i \in M \).

The decision variables for the model are as follows:

\[
X_{ijk} = \begin{cases} 
1, & \text{if Job } j \text{ precedes Job } k \text{ on Machine } i \\
0, & \text{otherwise,}
\end{cases}
\]

\[
Q_{ij} = \text{Quantity of Job } j \text{ that processed on Machine } i,
\]

\[
C_{ij} = \text{Completion time of Job } j \text{ at Machine } i, \quad C_{\text{max}} = \text{Maximum completion time.}
\]

The objective function is

\[
\text{Minimize } C_{\text{max}}.
\]

And the constraints are

\[
\sum_{i \in M} \sum_{j \in [0\cup N]} X_{ijk} \leq B[k], \quad \forall k \in N,
\]

\[
\sum_{i \in M} \sum_{j \in [0\cup N]} X_{ijk} \geq 1, \quad \forall k \in N,
\]

\[
Q_{ik} \leq V \sum_{j \in [0\cup N]} X_{ijk}, \quad \forall i \in M, \quad \forall k \in N,
\]

\[
\sum_{i \in M} Q_{ij} = q^p, \quad \forall j \in N,
\]

\[
\sum_{i \in M} Q_{i0} = 0,
\]

\[
\sum_{k \in N} X_{ijk} \leq 1, \quad \forall j \in N, \quad \forall i \in M,
\]

\[
\sum_{k \in N} X_{ik0} \leq 1, \quad \forall i \in M,
\]

\[
\sum_{h \in [0\cup N]} X_{hjk} \geq X_{ijk}, \quad \forall j, k \in N, \quad j \neq k, \quad \forall i \in M,
\]

\[
C_{ik} + V \left(1 - X_{ijk}\right) \geq C_{ij} + S_{ijk} + D_{ik} \ast Q_{ik}, \quad \forall j \in [0 \cup N], \quad \forall k \in N, \quad j \neq k, \quad \forall i \in M,
\]
\[ C_{ij} = 0, \quad \forall i \in M, \]  
\[ C_{ij} \geq 0, \quad \forall j \in N, \quad \forall i \in M, \]  
\[ C_{\text{max}} \geq C_{ij}, \quad \forall j \in N, \quad i \in M, \]  
\[ X_{ijk} \in \{0, 1\}, \quad \forall j \in \{0\} \cup \{N\}, \quad k \in N, \quad j \neq k, \quad \forall i \in M \]  
\[ Q_{ij} \geq 0 \text{ and Integer}, \quad \forall j \in N, \quad \forall i \in M. \]  

The objective (2) is to minimize the maximum completion time or makespan. Constraints sets (3) and (4) ensure that the number of subjobs will be between 1 and the maximum number of related subjobs. The usage of dummy jobs 0 as \( X_{ijk} \) indicates that \( k \) is the first job on Machine, Constraint set (5) is to control the production quantities. If Job \( j \) is processing on Machine, then production quantity \( Q_{ik} \) exists; otherwise, production quantity is zero. Constraint set (6) ensures that the total subjob quantity of the each job is equal to a constant number \( d^p \). Difference between unit processing times of each job will be calculated by the formulation of \( D_{ij} = p_j/q^p \) which will be used in constraint set (11). Constraints set (7) defines total production quantities as 0 for dummy jobs. Constraints set (8) prevents subjobs on the same machine. Constraints set (9) specify that not more than one subjob can be scheduled first at each machine. With set (10) we ensure that jobs are properly linked in machine: if a given Job, is processed on a given Machine, a predecessor \( h \) must exist on the same machine. Constraint set (11) is to control the completion times of the jobs at the machines. If Job, is assigned to Machine, after Job, \( X_{ijk} \) will be equal to 1. In this situation the completion time of \( k, C_{ik} \) must be greater than the completion time of \( j, C_{ij} \) plus the setup time between \( j \) and \( k \) and processing time of \( k \). The processing time of \( k \) is computed by multiplying unit processing time \( D_{ik} \) and \( Q_{k} \) which is production quantity of Job, on Machine, If \( X_{ijk} \) is equal to 0, then the big constant \( V \) renders the constraint redundant. Sets (12) and (13) define completion times as 0 for dummy jobs and nonnegative for regular jobs, respectively. Set (14) defines the maximum completion time. Set (15) defines binary variables. Finally, set (16) defines integer variables.

Optimal solution for the problem can be obtained by solving this MIP model using a solver. We coded this model using MPL and CPLEX II.0 is used as a solver. The model can solve this problem for 2-machine 6-job problem. In which all the jobs can be split into maximum 3 subjobs. But, for the 4-machine 6-job problem the model cannot give the optimal solution in reasonable elapsed time, one day. Because of this situation, additional constraints are added to this model to reduce the solution space. The new model, MIP2, that will be explained in Section 3.2 is identical with the GAspLA that will be explained in Section 4.

3.2. MIP2: The Quantities of Subjobs Are Equal. In this model, each subjob from one main job includes equal quantity of order. The model is also deciding the optimal number of subjobs. So, the content of this model is identical with the GAspLA that will be explained in Section 4.

To achieve equal subjob quantities in MIP2, constraint sets (17) and (18) will be added to the MIP1 that was explained in the previous subsection. Following logic constructs the constraints sets (17) and (18):

(i) If there is a splitting operation, the production quantities of Job, on Machine, and Machine, must be equal \( (Q_{ij} = Q_{ij}) \).

(ii) If there is splitting, then \( |Q_{ij} - Q_{ij}| = 0 \) must be satisfied.

(iii) Splitting condition is added to the inequality of \( |Q_{ij} - Q_{ij}| \leq \varepsilon \).

Consider

\[ Q_{ij} - Q_{ij} \leq 0.1 + V \ast \left( 2 - \sum_{h \in \{0\} \cup \{N\}} X_{hij} - \sum_{h \in \{0\} \cup \{N\}} X_{hij} \right) \]
\[ \forall i, t \in M, \quad i \neq t, \quad \forall j \in N, \]  
\[ (17) \]
\[ Q_{ij} - Q_{ij} \geq -0.1 - V \ast \left( 2 - \sum_{h \in \{0\} \cup \{N\}} X_{hij} - \sum_{h \in \{0\} \cup \{N\}} X_{hij} \right) \]
\[ \forall i, t \in M, \quad i \neq t, \quad \forall j \in N. \]  
\[ (18) \]

The quantities of subjobs \( (Q_{ij} \) and \( Q_{ij} \)) are integer, because of this constraint, \( \varepsilon \) or 0.1 difference between subjobs will lose the importance and the constraint will be provided. Otherwise, the big constant \( V \) renders the constraint redundant.

We coded this model using MPL and CPLEX II.0 is used to solve this model. While the model can solve the problem with 2 machines and 6 jobs, the 4-machine 6-job problem could not be solved even in one day. This circumstance has forced us to develop new model, MIP3, that is explained in Section 3.3.

3.3. MIP3: The Quantities of Subjobs Are Determined. In MIP3, the numbers of subjobs are determined before the model start running. To become close to the GAspLA, the numbers of subjobs are set randomly between 1 and the maximum number of subjobs for each individual jobs. The details about the model are as follows.
The decision variables for the model are as follows:

\[ X_{ijrks} = \begin{cases} 1, & \text{if } j r \text{ is } k \text{th subjob of Job}_j \text{ is precedes } k s \text{ which is } s \text{th subjob of } \text{Job}_k \text{ on Machine}_i \\ 0, & \text{otherwise,} \end{cases} \]  

(19)

\[ C_{ijr} = \text{Completion time of sub job } j r \text{ on Machine}_i, \]

\[ C_{\max} = \text{Maximum Completion Time.} \]

The objective function is

\[ \text{Minimize } C_{\max}. \]  

(20)

The constraints are

\[ \sum_{i \in M} \sum_{j \in \{0\} \cup \{N\}} \sum_{r \in B} X_{ijrks} = 1, \quad \forall k \in N, \forall s \leq B[k], \]  

(21)

\[ \sum_{i \in M} \sum_{k \in N} \sum_{s \in B} X_{ijrks} \leq 1, \quad \forall j \in N, \forall r \leq B[j], \]  

(22)

\[ \sum_{r \in B} \sum_{k \in N} \sum_{s \in B} X_{ijrks} \leq 1, \quad \forall i \in M, \]  

(23)

\[ C_{iks} + V \left( 1 - X_{ijrks} \right) \geq C_{ijr} + S_{ijk} + \frac{P_{ik}}{B[k]} \]  

\[ \forall i \in M, \forall j \in \{0\} \cup \{N\}, \forall r \leq B[j], \]  

(24)

\[ C_{i0r} = 0 \quad \forall i \in M, \forall r \leq B[j], \]  

(25)

\[ C_{ijr} \geq 0 \quad \forall i \in M, \forall j \in N, \forall r \leq B[j], \]  

(26)

\[ C_{\max} \geq C_{ijr} \quad \forall i \in M, \forall j \in N, \forall r \leq B[j], \]  

(27)

\[ X_{ijrks} \in \{0, 1\} \]  

\[ \forall j \in \{0\} \cup \{N\}, \forall r \leq B[j], \]  

(28)

\[ \forall k \in N, \forall s \leq B[k], \]  

(29)

\[ \forall i \in M, j = k \implies r \neq s. \]

The usage of dummy job 0 as subjob is assigned to one machine and has one predecessor. Constraints (22) make sure that the maximum number of successors of every subjob is one. Constraints (23) specify that not more than one subjob can be scheduled first at each machine. Constraint set (24) ensures that jobs are properly linked in machine. If subjob jr is processed on a given Machine, a predecessor hm must be processed on the same machine. Constraints (25) are used to calculate and control completion times. If a subjob ks is assigned to Machine, after subjob jr, then \( X_{ijrks} = 1 \). In that condition \( C_{iks} \) must be greater than or equal to the completion time of jr and \( C_{ijr} \), plus the setup time between Jobj and Jobk, and the processing time of subjob ks which is calculated by processing time of Jobk on Machinei, \( P_{ik} \), divided by the number of subjobs of Jobk, \( B[k] \). If \( X_{ijrks} = 0 \), then the big constant \( V \) renders the constraint redundant. Constraints (26) state that the completion time for the dummy job 0 is zero and constraints (27) ensure that completion times are nonnegative for regular jobs. Set (28) defines the maximum completion time. Set (29) defines the binary variables.

MIP3’s performance is better than the others and gave optimal solution to many of the problem sets which will be explained in the computational results. We coded this model using MPL and to solve the models, CPLEX 11.0 is used as a solver.

4. GAsP-LA: Hybrid Genetic-Local Search Algorithm for Job Splitting Property

4.1. Genetic Algorithm for Job Splitting Property. The genetic algorithm (GA) is a search technique based on the principles of genetics and natural selection. One of the important issues is the genetic representation (string of symbols) of each solution in a population. The string is referred to as chromosome and the symbols as genes. After generation of initial population and determination of the fitness function values for each chromosome, the GA manipulates the selection process by operations such as reproduction, crossover, and mutation. The algorithm was introduced in the 1970s by Holland [30]. As the searching technique of genetic algorithms (GAs) [31] became popular in the mid-1980s, many researchers started to apply this heuristic to scheduling problems.

The encoding must be designed to utilize the algorithm’s ability to transfer information among chromosome strings efficiently and effectively [31]. To qualify our encoding scheme, the permutation of jobs in this work is shown through random keys. Each job has a random number between 0 and 1. These random keys show the relative order of the jobs for each machine. In addition, each chromosome also carries the number of subjobs. Detailed description of developed genetic algorithm has been reported in the following subsections.

The proposed local search algorithm which will be explained in Section 4.2 is inserted in fitness function calculation process of genetic algorithm. The proposed hybrid genetic-local search algorithms are coded in C# language according to proposed methods that were explained in the next subsections.
4.1.1. Encoding Scheme. A chromosome is presented as a string of random keys. Figure 1 illustrates a sample chromosome. We would like to note that this chromosome structure is used in our previous study [28] which proposed a genetic algorithm without local search. In the first section of the chromosome, the string contains \( n \) (number of jobs) sections. Each section is further divided into \( m \) (number of machines) sections. Each section for machines is divided into subsections (genes). The number of subsections for related machine is determined according to maximum number of subjobs for related job. Sequence of jobs will be determined by random key numbers (generated between 0 and 1) of each gene. In the second section, the string contains \( n \) (number of jobs) sections. For each section, a random number will be generated between one and the max numbers of subjobs for each job to determine the number of subjobs. The chromosome structure of the example is in Figure 1. For this chromosome structure, we have 2 machines and 3 jobs. Maximum numbers of subjobs for Job 1, Job 2, and Job 3 are 3, 2, and 1, respectively. For Job 1, because of number of subjobs as the value of 2 in second section of chromosome, the smallest two random numbers will be selected from first section of the chromosome among all the numbers generated for Job 1. Selected values (0.2 and 0.3 for Job 1) are given in bold in Figure 1. It's clear that the first and second subjobs of Job 1 will be processed on Machine 1. Similarly, number of subjobs is 2 for Job 2 and they will be processed on Machine 1 and Machine 2. Number of subjobs is 1 for Job 3 and it will be processed on Machine 2.

4.1.2. Fitness Function \( (C_{\text{max}}) \). As mentioned earlier, there are machine-dependent processing times and machine and job sequence-dependent setup times:

(i) \( p_{ij} \): processing time of Job \( j \), \( j \in N \) at Machine \( i \), \( i \in M \).

(ii) \( S_{ijk} \): machine based sequence dependent setup time on Machine \( i \), \( i \in M \) when processing Job \( k \) \( k \in N \) after having processed Job \( j \), \( j \in N \).

In case of considering three jobs, two machines scheduling problem, for example, Table 1 is the processing times for Machine 1 and Machine 2, Tables 2 and 3 are setup times of Machine 1 and Machine 2, respectively, for the mentioned jobs. According to chromosome that was shown in Figure 1, Machine 1 will process two subjobs of Job 1 (Subjob 1,1,2) and one subjob of Job 2 (Subjob 2). Machine 2 will process one subjob of Job 2 (Subjob 2) and one subjob of Job 3 (Subjob 3). The sequence of jobs on Machine 1 will be determined according to random key numbers (0.20, 0.26, and 0.30). Increasing arrangement of these random numbers will designate Machine 1’s job sequence. So, sequence of subjobs is Subjob 1,1, Subjob 2 and Subjob 2 on Machine 1. If a job is split into subjobs, processing time of required subjob on selected machine can be calculated by dividing a job’s processing time on the related machine to the number of subjobs. Completion time of Machine 1 is 130 and completion time of Machine 2 is 78 according to \( p_{ij} \) and \( S_{ijk} \) values. So, \( C_{\text{max}} \) will be 130. Mentioned chromosome’s schedule and makespan value can be seen in Figure 2. Population size will determine number of different chromosomes and \( C_{\text{max}} \) values.

4.1.3. Genetic Operators. After generating initial population, selection, crossover, and mutation will be iteratively used to search for the best solution.

(i) Selection: chromosomes will be selected into the mating pool based on random selection method [32]. In this method, moms and dads are randomly chosen from population.

(ii) Crossover: the crossover operator, which was applied according to value of crossover rate, \( P_c \), is a method for sharing information between chromosomes. Single point crossover will be used as crossover operator in our algorithm. It randomly chooses the crossing point and exchanges the genes between two parents in order to create offsprings. Crossover operator will be used for the first and second sections separately. Figure 3 illustrates this operation. In Figure 3, for the first section the crossing point is selected randomly at position seven. The child is created as follows. First, Parent 1 passes its genes that are on the left of the crossover point (position seven) to Offspring 1. In a similar manner, Parent 2 passes its genes to the left of the same crossover point to Offspring 2. Next, the genes on the right of the crossover point of Parent 2 (Parent 2) are copied to Offspring 2 (Offspring 2). For the second section, crossing point is selected at position one randomly. Same procedure to create the offsprings is applied for the second section of the chromosome.

(iii) Mutation: the mutation operator is used to prevent converging to a local optimum. In our algorithm, a mutation is performed as follows. Chromosomes that mutation operation will be applied to are randomly selected from the population according to mutation

| \( P_{ij}P_{ij} \) | Job 1 | Job 2 | Job 3 |
|----------------|--------|--------|--------|
| Machine 1      | 90     | 60     | 30     |
| Machine 2      | 95     | 70     | 40     |

| \( S_{ijk}S_{ijk} \) | Job 1 | Job 2 | Job 3 |
|----------------|--------|--------|--------|
| Job 1          | —      | 8      | 6      |
| Job 2          | 2      | —      | 4      |
| Job 3          | 3      | 7      | —      |

| \( S_{ijk}S_{ijk} \) | Job 1 | Job 2 | Job 3 |
|----------------|--------|--------|--------|
| Job 1          | —      | 2      | 6      |
| Job 2          | 4      | —      | 3      |
| Job 3          | 2      | 1      | —      |
rate ($P_m$). The value of the randomly selected gene of the chromosome will be replaced with a new random number. This operation will be applied to all selected chromosomes. In this way, chromosomes with new schedules and makespan values can be obtained. The fitness value may be better or worse or may not be changed after applying this operator. For the example in Figure 4 if randomly selected chromosome is Parent, and randomly selected gene's key number 0.54 would be changed to the key number 0.15 which is also randomly generated for the mutation operation, the sequence on Machine, and Machine, will be altered. Because selected values for Job, will be 0.20 and 0.15 (are given in bold in Figure 4), Machine, will process one subjob of Job, (Subjob), and one subjob of Job, (Subjob). Machine, will process one subjob of Job, (Subjob), one subjob of Job, (Subjob), and one subjob of Job, (Subjob). This will be a cause of new makespan value of 127.5 which is better than the previous one.

4.2. Local Search. Integrating one of the local search techniques within GA will generally generate more competitive results. The integrated dominance properties method was proposed by Chang and Chen [20]. In this method, the decision to exchange two jobs (intramachine or intermachines) is made according to the result of calculations, which are described in detail in the cited paper. While adapting this method to our GA, we designed a local search method based on integrated dominance properties. Our aim is searching all possible alternatives in chromosome in order to get better processing and setup times for each job. In the proposed local search method, the following three disparities were created as compared with integrated dominance properties method.

(1) Job interchange: we only used setup times instead of adjusted processing times while interchanging two jobs that are processed on the same machine.

(2) Job exchange: while the jobs under consideration are not on the same machine, setup times and processing times are both considered in the calculations.

(3) During the job interchanges or exchanges on chromosome, also the random numbers must be redesigned. So, after the decision to interchange or exchange jobs is made, random numbers must be redesigned to achieve the required sequence. In order to change minimum number of random numbers on the chromosome, a new approach is generated. This process (calibration of random numbers) is also managed and coded in the proposed local search method.

All of the situations for local search have been considered and written in pseudocode, which is summarized in the Appendix. The proposed intermachine (interchange) and intramachine (exchange) job changes will be explained in Section 4.2.1 and the calibration of random numbers will be explained in Section 4.2.2. The notation for the pseudocode and explanations of local search are as follows.

$g$: number of selected genes from the chromosome, which is ordered by machine and then by job sequence. Each individual gene contains machine and job features. Table 4 explains the sample gene structure for an example with seven jobs and two machines. The structure also indicates the sequence of jobs for each machine.

(i) $i$ and $j$: comparing gene numbers to decide whether the jobs in these genes should be exchanged.

(ii) Gene($i$)$_{Machine}$ indicates the machine number on $i$th gene. According to the structure shown in Table 4, the machine number for 1st gene (Gene(1)$_{Machine}$) is 0.

---

**Table 4:** Sample Gene Structure

| Job | Machine 1 | Machine 2 |
|-----|-----------|-----------|
| 1   | Subjob1   | Subjob2   |
| 2   | Subjob3   | Subjob4   |

**Figure 1:** Representation of a chromosome in the GA.

**Figure 2:** Resulting schedule and $C_{max}$ for chromosome in Figure 1.
Crossover point for first section is seven
Crossover point for second section is one

**Figure 3:** Crossover operation: two parents mate in order to produce two offsprings.

**Figure 4:** Mutation operation.

(iii) Gene\(i\)\(_{job}\) indicates the job number on \(i\)th gene. According to the structure shown in Table 4, the job number for 1st gene (Gene\(1\)\(_{job}\)) is 4.

(iv) \(ST_{Mach(i)Job(j)k}\): setup time on Gene\(i\)\(_{Machine}\) between Gene\(j\)\(_{job}\) and Gene\(k\)\(_{job}\).

(v) \(PT_{Mach(i),job(j)}\): processing time of Gene\(j\)\(_{job}\) on Gene\(i\)\(_{Machine}\)

(vi) \(CT_{Gene(i)Machine}\): completion time of Gene\(i\)\(_{Machine}\)

### 4.2.1. Interchange and Exchange of Jobs

(i) **Interchange of jobs:** there are two cases to be considered within the intermachine interchange, they are adjacent jobs interchange (see Figure 5) and nonadjacent jobs interchange (see Figure 6).

(I) **Adjacent interchange:** in order to define the algorithm better, adjacent jobs interchange case is also demonstrated in Figure 5. Arrows on Figure 5 show setup time (among stated genes) differences between after interchange and before interchange situations. In Figure 5 and intermachine, adjacent jobs interchange section of pseudocode, steps are as follows:

1. if Gene\((i-1)\)\(_{job}\) precedes Gene\(i\)\(_{job}\) on before interchange section of Figure 5, “\(a\)” represents sum of setup time differences between after interchange and before interchange situations for stated blue arrows.
2. If Gene\(i\)\(_{job}\) is the first job on the machine on before interchange section of Figure 5, “\(b\)” represents sum of setup time differences between after interchange and before interchange situations for stated blue arrows.

(3) Either “\(a\)” or “\(b\)” must be equal to 0.

(4) If Gene\(j\)\(_{job}\) precedes Gene\((j+1)\)\(_{job}\) on Figure 5, “\(c\)” represents setup time differences between after interchange and before interchange situations for stated red arrow.

(5) If the sum of \(a\), \(b\), and \(c\) is smaller than 0, it means that better completion time is obtained. Let us interchange Gene\(i\)\(_{job}\) and Gene\(j\)\(_{job}\) on the schedule.

(6) Then calibrate random key numbers. In Section “4.2.2”, item “i” is description of intermachine interchange.

(II) **Nonadjacent interchange:** this case is shown in Figure 6. The steps to decide whether interchange of Gene\(i\)\(_{job}\) and Gene\(j\)\(_{job}\) on the schedule are similar with adjacent interchange. During the calculation of “\(a\)” and “\(b\)”, three blue arrows must be sum which are shown on Figure 6. This is caused by new setup time differences coming into existence between after interchange and before interchange situations. The remaining steps are the same as adjacent interchange situation.

(ii) **Intramachine exchanging** is shown on Figure 7: in Figure 7 and intramachine exchanging section of pseudocode steps are as follows:

1. calculate completion times of Gene\(i\)\(_{Machine}\) and Gene\(j\)\(_{Machine}\) before exchange. “\(D1\)” represents maximum value of these two values.
2. If Gene\((i-1)\)\(_{job}\) precedes Gene\(i\)\(_{job}\) on Gene\(i\)\(_{Machine}\) which was shown before exchange section of Figure 7,
Before interchange
Gene\( i \)\text{Machine} = Gene\( j \)\text{Machine}

After interchange
Gene\( i \)\text{Machine} = Gene\( j \)\text{Machine}

\[ a \Rightarrow (ST\text{Mach}(i)\text{Job}(i,j) - ST\text{Mach}(i)\text{Job}(i-1,j)) + (ST\text{Mach}(j)\text{Job}(j,i) - ST\text{Mach}(j)\text{Job}(j-1,i)) \]

\[ b \Rightarrow (ST\text{Mach}(i)\text{Job}(j,j) - ST\text{Mach}(i)\text{Job}(i,i)) \]

\[ c \Rightarrow ST\text{Mach}(j)\text{Job}(j+1) - ST\text{Mach}(j)\text{Job}(j+1) \]

**Figure 5:** Intermachine interchange and adjacent jobs situation.

Before interchange
Gene\( i \)\text{Machine} = Gene\( j \)\text{Machine}

After interchange
Gene\( i \)\text{Machine} = Gene\( j \)\text{Machine}

\[ D_1 = \text{Max} (CT\text{Gene}(i)\text{Machine} ; CT\text{Gene}(j)\text{Machine}) \]

\[ a_1 = ST\text{B}_2 - ST\text{B}_1 \]

\[ b_1 = ST\text{Y}_2 - ST\text{Y}_1 \]

\[ a_2 = ST\text{W}_2 - ST\text{W}_1 \]

\[ b_2 = ST\text{P}_2 - ST\text{P}_1 \]

**Figure 6:** Intermachine interchange and nonadjacent jobs situation.

Before exchange
Gene\( i \)\text{Machine} = Gene\( j \)\text{Machine}

After exchange
Gene\( i \)\text{Machine} = Gene\( j \)\text{Machine}

\[ D_2 = \text{Max} (CT\text{Gene}(i)\text{Machine} + PT\text{Mach}(i)\text{Job}(i) - PT\text{Mach}(j)\text{Job}(j), a_1 + a_2; CT\text{Gene}(j)\text{Machine} + PT\text{Mach}(j)\text{Job}(j) + b_1 + b_2) \]

**Figure 7:** Intramachine exchange (exchange of Gene\( i \)\text{Job} and Gene\( j \)\text{Job} on the schedule).

**Figure 8:** An example for calibration of random numbers on the chromosome for intermachine interchanges.
“a1” represents setup time differences between after interchange and before interchange situations for stated blue rectangles (STB1, STB2). (a1 = STB2 − STB1.) (If Gene(i)job is the first job on Gene(i)machine, second “a1” of pseudocode represents sum of setup time differences between after interchange and before interchange situations.)

(3) If Gene(i)job precedes Gene(i + 1)job on Gene(i)machine which was shown before exchange section of Figure 7, “a2” represents setup time differences between after interchange and before interchange situations for stated white rectangles (ST1W, ST2W). (a2 = ST2W − ST1W.)

(4) If Gene(j − 1)job precedes Gene(j)job on Gene(j)machine which was shown before exchange section of Figure 7, “b1” represents setup time differences between after interchange and before interchange situations for stated yellow rectangles (STY1, STY2). (b1 = ST2Y − ST1Y.) (If Gene(j)job is the first job on Gene(j)machine, second “b1” of pseudocode represents sum of setup time differences between after interchange and before interchange situations.)

(5) If Gene(j)job precedes Gene(j + 1)job on Gene(j)machine which was shown before exchange section of Figure 7, “b2” represents setup time differences between after interchange and before interchange situations for stated pink rectangles. (STP1, STP2,) (b2 = STP2 − STP1.)

(6) Calculate new completion times of Gene(i)machine and Gene(j)machine after possible exchange. “D2” represents maximum value of these two values. Formulation the computation of D2 is shown in Figure 7.

(7) If D2 is smaller than D1 it means that better completion time is obtained. Let us exchange Gene(i)job and Gene(j)job on the schedule.

(8) Then calibrate random key numbers. In Section “4.2.2”, item “ii” is description of intramachine exchange.

4.2.2. Calibration of Random Numbers on the Chromosome. If we decided to change Gene(i)job and Gene(j)job, we need to calibrate the chromosome to recognize the new situation. To recognize the job changes by chromosome, the following configurations were also integrated in the code. During the integration process, the places of random numbers are considered. JobA is split into 2 subjobs, JobB is not split, and JobC is split into 2 subjobs for the considered chromosome. The number of subjobs has been determined randomly as pointed out before. Chromosomes of Figures 8 and 9 do not show the section of number of subjobs.

(i) Intermachine interchange: Figure 8 shows before and after interchange of jobs on the same machine. Selected genes from chromosome and nonselected genes from chromosome constitute a chromosome. The realized interchanges are shown in red letter on the after interchange section of Figure 8. The steps of interchanging Gene(i)job and Gene(j)job for the example of Figure 8 are the following.

(I) Random numbers of the jobs will be exchanged (0.40 for Gene(i)job → JobA and 0.62 for Gene(j)job → JobC) will be changed to 0.40 for Gene(i)job → JobC and 0.62 for Gene(j)job → JobA).

(II) 0.40 is smaller than 0.62. Thus, after the interchange there is no need to change the other random numbers for JobC. Because the current situation guarantees JobC to select Machine0.

(III) 0.62 is bigger than 0.40. Thus, after the interchange of the other random numbers for JobA those smaller than 0.62 inside the “nonselected genes from chromosome” must be changed. For example, 0.41 is smaller than 0.62. A new random number which is bigger than 0.62 will be randomly generated (0.96) and changed with 0.41. Another random number is 0.61 inside the “nonselected genes from chromosome” for JobA which is smaller than 0.62. A new random number which is bigger than 0.62 will be randomly generated (0.75) and changed with 0.61. Thus, JobA satisfies selecting Machine0.

(ii) Intramachine exchange: Figure 9 shows before and after exchange of jobs on two different machines. The realized
exchanges are shown in red letter on the after exchange section of Figure 9. The steps of exchanging Gene(i)_{job} and Gene(j)_{job} for the example of Figure 9 are as follows. Value(i) is a maximum random number value for Gene(j)_{job} on Gene(i)_{Machine}. Similarly, Value(j) is a maximum random number value for Gene(i)_{job} on Gene(j)_{Machine}.

(I) Random numbers for Gene(i) and Value(i) are exchanged. And also random numbers for Gene(j) and Value(j) are exchanged.

(II) After the exchange it must be checked if there is a random number inside the "nonselected genes from chromosome" for Gene(j)_{job} → Job_A that is smaller than random number of Gene(j) which is equal to 0.84 or other selected value of Job_A which is equal to 0.34. For example, 0.41 is smaller than 0.84. Another random number which is bigger than 0.84 will be randomly generated (0.95) and changed with 0.41.

(III) Between selected genes from chromosome and nonselected genes from chromosome, jobs for Gene(i) and Value(i) are exchanged. And also jobs for Gene(j) and Value(j) are exchanged.

(IV) It must be also checked if there is a random number inside the "nonselected genes from chromosome" for Gene(i)_{job} → Job_C that is smaller than random number of Gene(i) which is equal to 0.40 or other selected value of Job_C which is equal to 0.62. For example, 0.61 is smaller than 0.62. Another random number which is bigger than 0.62 will be randomly generated (0.87) and changed with 0.61.

5. Implementation: GAspLA-to-MIP Method (GAspLAMIP)

In this section, we propose GAspLA based mix-integer programming model for the mentioned problem. This implementation will help us to reduce the elapsed time of MIP3. And this implementation will also show that our hybrid genetic-local search algorithm (GAspLA) gives optimal results for the studied datasets which will be discussed in Section 6.

As we mentioned, we obtain final scheduling using GAspLA. Then, according to the sequencing information of subjobs on each machine, we set binary variable values in the MIP3 formulation as initial values. For example, according to Table 5, there are 4 machines and 6 jobs. Each machine’s first job must be dummy job (j : 0, r : 1). According to this information, first row of Table 5 indicates that on Machine_1, subjob_{b1} will be processed. And according to second and third rows of Table 5, Machine_2’s sequence will be subjob_{b1} and subjob_{b1}. The schedules of the other machines can be interpreted similarly.

In order to reduce the search space of MIP3, the result schedule of GAspLA will be used. Therefore, running MIP3 with the initial solution of GAspLA will help us to accelerate to reach the solution. In Section 3.3, MIP3 was proposed and in Section 4 GAspLA was proposed. The steps for GAspLAMIP are as follows.

6. Experimental Design and Computational Results

The proposed algorithm is coded in C# language and run on computer with an Intel Core i7-3612QM processor running at 2.10 GHz. Test samples are provided by Scheduling Research 2005 [4].

The parameter configuration of GALA that was proposed by Yilmaz Eroglu et al. [21], which is the foundation of GAspLA, is done by the Design of Experiment (DoE). The scheduling environment difference between GALA and GAspLA is job splitting property. We will make comparison also with GALA and to avoid advantages of parameter configuration, we run GALA, GAspLA, and GAspLAMIP with the same parameters which are obtained by DoE approach that is detailed in next subsection.

6.1. Experimental Results and Parameter Values. Makespan values and elapsed times were registered for the selected 4-machine 20-job problem structure for every 100 generations up to 1000 generations to determine the optimal number of generations. Figure 10 provides the average makespan values and elapsed times of five replications for the analyzed generations with the selected parameters. Figure 10 shows that after 500 generations, there is no improvement in the makespan. Thus, the number of generations can be set to 500.

The 4-machine 20-job problem structure was chosen to determine the optimal parameter setting by DoE. The levels of the three factors are listed in Table 6. Five replications were conducted for each combination, and the makespan values were calculated by running the program until 500 generations had elapsed. In Table 7, the estimates obtained through the regression analysis are shown, including significant factors (P-value ≤ 0.05). The results indicate that the probability of crossover (Pc) and probability of mutation (Pm) are significant, with low P values shown in Table 7. The interaction plot in Figure 11 and the low P value of Pc * Pm in Table 7 indicate that there is a significant interaction between Pc and Pm. According to Figures 11 and 12, when Pc is set to 1 and Pm is set to 0.2, the algorithm will yield a better solution quality. The population size (Psize) is set to 100 according to the main effects plot in Figure 12.

The obtained results are evaluated in two subsections. In the first subsection, the results of MIP that was proposed...
Average makespan values

Number of generations

Elapsed time (s)

Figure 10: Average makespan values and elapsed times for the determined number of generations.

Figure 11: Interaction plot of \( P_c \) and \( P_m \).

Table 4: Sample gene structure for an example that contains seven jobs and two machines.

| Gene number | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|-------------|---|---|---|---|---|---|---|
| Machine number | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| Job number | 3 | 4 | 6 | 1 | 0 | 2 | 5 |

Table 5: Initial values for the binary variable set.

| Machine number \( i \) | Subjob sequence | Value of binary variable \( X_{ijk}s \) |
|------------------------|-----------------|---------------------------------|
| 1                      | 0 1 3 1         | 1                               |
| 2                      | 0 1 1 1         | 1                               |
| 2                      | 1 1 4 1         | 1                               |
| 3                      | 0 1 6 1         | 1                               |
| 3                      | 6 1 5 1         | 1                               |
| 4                      | 0 1 2 1         | 1                               |
| 4                      | 2 1 6 2         | 1                               |

Table 6: Parameter values used to assess the proposed genetic algorithm.

| Parameter | Value |
|-----------|-------|
| Population size \( (P_{size}) \) | 80; 100 |
| Probability of crossover \( (P_c) \) | 0.5; 1.0 |
| Probability of mutation \( (P_m) \) | 0.2; 0.5 |

Table 7: Parameter estimates.

| Term | Effect | Coef. | SE coef. | \( T \) | \( P \) |
|------|--------|-------|----------|--------|--------|
| Constant | 606.025 | 0.2789 | 2172.53 | 0.000  |
| \( P_{size} \) | −0.450 | −0.225 | 0.2789 | −0.81 | 0.426  |
| \( P_c \) | −2.450 | −1.225 | 0.2789 | −4.39 | 0.000  |
| \( P_m \) | 1.350 | 0.675 | 0.2789 | 2.42 | 0.021  |
| \( P_{size} \) \( * \) \( P_c \) | 0.050 | 0.025 | 0.2789 | 0.09 | 0.929  |
| \( P_{size} \) \( * \) \( P_m \) | −0.150 | −0.075 | 0.2789 | −0.27 | 0.790  |
| \( P_c \) \( * \) \( P_m \) | −1.150 | −0.575 | 0.2789 | −2.06 | 0.047  |
| \( P_{size} \) \( * \) \( P_c \) \( * \) \( P_m \) | 0.750 | 0.375 | 0.2789 | 1.34 | 0.188  |

by Rabadi et al. [18] and the results of GALA which was proposed by Yilmaz Eroglu et al. [21] will be analyzed for nonjob splitting situation. For the job splitting situation, the results of MIP3 that was proposed in Section 3.3, the results of GAspLA that was proposed in Section 4, and the results of implementation (GAspLAMIP) that was proposed in Section 5 will be explicated in Section 6.2. In Section 6.3, detailed results of solving some literature problems using the proposed GAspLA are submitted.

6.2. The Results of MIP, GAspLA and GAspLAMIP. There are 15 different problem instances in the dataset SchedulingResearch 2005 that was taken from the literature [4]. For the combinations which were shown on the Table 8, the first instances of these datasets are used and the following cases are studied and reported.

(i) For the nonjob splitting situation, MIP model which was proposed by Rabadi et al. [18] is coded in MPL and solved using CPLEX 11.0. The makespan values and elapsed times are reported in the nonjob splitting situation, MIP0 section of Table 8.

(ii) For the nonjob splitting situation, the makespan values and elapsed times of the GALA algorithm which was proposed by Yilmaz Eroglu et al. [21] are reported in the nonjob splitting situation, GALA section of Table 8.
Main effects plot (data means) for makespan

![Main effects plot](image)

**Figure 12:** Main effects plot of the three factors.

**Table 8:** The comparison of solution methods.

| # of Job | # of Machine | Non job splitting situation | Job splitting situation |
|----------|--------------|------------------------------|-------------------------|
|          |              | MIP0 Makespan Value | Elapsed Time (second) | GALA Makespan Value | Elapsed Time (second) | MIP3 Makespan Value | Elapsed Time (second) | GAspLA Makespan Value | Elapsed Time (second) | GAspLAMIP Makespan Value | Elapsed Time (second) |
| 6        | 2            | 390 3                    | 390 3                  | 390 545             | 390 9                 | 390 14               |
| 6        | 4            | 235 3                    | 235 4                  | 234 1106            | 234 13                | 234 16               |
| 7        | 2            | 484 23                   | 484 4                  | 470.5 35172         | 470.5 13              | 470.5 204            |
| 7        | 4            | 258 3                    | 258 4                  | 245 7200            | 245 22                | 245 26               |
| 8        | 2            | 494 232                  | 494 5                  | — >86400            | 494 16                | 494 245              |
| 8        | 4            | 264 6                    | 264 7                  | 264 80096           | 264 27                | 264 29               |
| 9        | 2            | 627 2120                 | 627 6                  | — >86400            | 627 19                | 627 45394            |
| 9        | 4            | 345 6345                 | 345 9                  | — >86400            | 331 30                | 331 38376            |
| 10       | 2            | 647 28144                | 647 7                  | — >86400            | 647 28                | 647 29842            |
| 10       | 4            | 360 58626                | 360 11                 | — >86400            | — 352.5 40            | — >86400             |

(iii) For the job splitting situation, the proposed model which is discussed in Section 3.3 was coded in MPL and solved using CPLEX 11.0. The makespan values and elapsed times are reported in job splitting situation, MIP3 section of Table 8.

(iv) For the job splitting situation, the makespan values and elapsed times of the proposed GAspLA algorithm are reported in the job splitting situation, GAspLA section of Table 8.

(v) For the job splitting situation, the makespan values and elapsed times of the GAspLAMIP which is discussed in Section 5 are reported in job splitting situation, GAspLAMIP section of Table 8.

The results of the models whose elapsed times exceed one day (86400 seconds) are not reported in Table 8. To point out the improvement through the proposed algorithm for the problem with job splitting, nonjob splitting situation results are also reported. If the analyzed problem has the job splitting property, then it is better to use the proposed algorithms including job splitting. The results on Table 8 can be interpreted as follows.

For the nonjob splitting situation, MIP0 model that was proposed by Rabadi et al. [18] found optimal solution for all the problem combinations in Table 8. The algorithm of GALA which was proposed by Yilmaz Eroglu et al. [21] also solved the problems on all these points in Table 8 with optimal values. For the job splitting situation the proposed MIP3 model which is introduced in Section 3.3, managed to find the optimal solutions for 5 combinations (6 jobs, 2 and 4 machines; 7 jobs, 2 and 4 machines; 8 jobs, 4 machines) in acceptable CPU time. The proposed algorithm of GAspLA
could manage to find the optimal results in a couple of seconds on these points. MIP3 and GAspLA algorithms for job splitting situation find the same or better results for the combinations as compared with the nonjob splitting situation. In addition, the implementation of GAspLAMIP which is performed by using GAspLA results as initial solution for MIP3 is solving 9 combinations (6 jobs, 2 and 4 machines; 7 jobs, 2 and 4 machines; 8 jobs, 2 and 4 machines; 9 jobs, 2 and 4 machines; 10 jobs, 2 machines) optimally in acceptable CPU time. The elapsed time of the GAspLAMIP is less than MIP3 as it can be seen in Table 8. The optimal makespan values of GAspLAMIP are the same with the result of GAspLA for the studied combinations. The results for these combinations demonstrated that the makespan values of GAspLA are close to the lower bounds. In the next subsection the performance of the GAspLA for the other combinations are shown in detail.

6.3. The Results of GAspLA for Some Literature Problems.
To test the proposed GAspLA algorithm, SchedulingResearch 2005 [4] dataset was used. ACO which was proposed by Arnaout et al. [19], GALA which was proposed by Yilmaz Eroglu et al. [21], and our new approach GAspLA were compared using the mentioned dataset.

There is no dataset for the problem of \(R_{mj}/S_{ijk}/C_{\max}\) with job splitting in the literature according to our knowledge. An original dataset did not contain number of splits. For that reason, number of splitting has been randomly generated between 1 and 3. For all instances, the processing and setup times that were uniformly distributed between 50 and 100 were called balanced datasets. For small instances, the combinations were 2 machines and 6, 7, 8, 9, 10, and 11 jobs, 4 machines and 6, 7, 8, 9, 10, and 11 jobs, 6 machines and 8, 9, 10, and 11 jobs, and 8 machines and 10 and 11 jobs. For large instances, the combinations were 4 machines and 20 jobs, 6 machines and 20 and 40 jobs, 8 machines and 20 and 60 jobs, and 10 machines and 20 and 40 jobs. Each machine-job combination was tested with 15 instances of the problem and the average deviation from a lower bound (LB) was calculated. The method for calculation of LB was explained in the paper of Arnaout et al. [19]. And the same LB values are used.

The percent deviation from lower bound is calculated as follows

\[
\text{Percent deviation from LB} = \frac{C_{\max,\text{Algorithm}} - LB}{LB} \times 100, \quad (30)
\]

where \(C_{\max,\text{Algorithm}}\) is the makespan value of the algorithm and LB is lower bound for the concerned instance.

Figure 13 summarizes the relative deviation of each algorithm from LB for small problem combination. It’s clear from Figure 13 that the GAspLA outperformed the others for the small problem sets. So if the problem can be split in subjobs, our new approach gives better results.

Figure 14 summarizes the relative deviation of each algorithm from LB for large problem combination. It’s clear from Figure 14 that the GAspLA outperformed the ACO on each combination. The GAspLA outperformed the GALA on 6 machines and 20 jobs and 8 machines and 20 jobs combinations. The GAspLA finds similar results with GALA on the other points. The number of splitting for GAspLA can be selected between 1 and 3 as pointed out before. The GAspLA decides not to split, so the number of splitting is 1, in the case of not finding better results. This is the main reason of obtaining similar results with the GALA, which is the nonjob splitting version of this study.

7. Conclusions and Future Direction of Research

The present paper presented mixed integer programming (MIP) models, genetic-local search algorithm (GAspLA), and an implementation of GAspLAMIP for the unrelated parallel machine scheduling problem with machine-dependent and sequence-dependent setup times and job splitting property to minimize the makespan. In this problem, it was assumed that a job can be split into a number of subjobs and these subjobs are processed on the unrelated parallel machines independently. Very little research appears to have been undertaken on this general model. The proposed hybrid genetic-local search algorithms are coded in C# language according to proposed methods.

The first contribution of this paper is capability of the developed algorithms to make splitting and scheduling simultaneously with variable number of subjobs, where, to the best of our knowledge, no work has been published on an algorithm with these properties. The proposed hybrid genetic-local search algorithm (GAspLA) is innovated in its use of random key numbers to determine the number of subjobs for each job and managing them to sequence on the machines. In order to overcome the problem of managing random key numbers according to the desired job sequence that might appear during local search operation which implemented in the genetic algorithm, some methods are developed. The developed algorithms satisfy the adaptation of results of local search into the genetic algorithms with minimum relocation operation of genes’ random key numbers. This is the second contribution of the paper. The third contribution of this paper is developed MIP models, which are coded in MPL and solved using CPLEX 11.0 solver. According to the best of our knowledge, there is no MIP approach for the problem type studied in this research. The results that are reported here indicate that the mentioned problem can be solved using the developed MIP models. Three mathematical models are presented on this topic. In the MIP1 and the MIP2, numbers of the subjobs will be determined by models between one and the maximum number of subjobs. In the MIP3, the numbers of subjobs will be determined randomly between one and maximum number of subjobs beforehand and will be given as an input data to the model. In the MIP1, the quantities of subjobs are not equal. In the MIP2 and MIP3 the quantities of subjobs are equal. The fourth contribution of this paper is an implementation of combining the results from GAspLA with the developed MIP formulation. This implementation (GAspLAMIP) also shows us the optimality of the results of
GAspLA for the studied combinations. We used Design of Experiment to select the parameter values for hybrid genetic-local search algorithms. The proposed methods are tested on a set of small and large problems taken from the literature (Scheduling Research 2005 [4]), and the computational results validate the effectiveness of the proposed algorithm.

The algorithms suggested in this paper may be useful for customer oriented market environments just like weaving process of textile industry that this study is originated from. And also drilling of printed circuit board (PCB) manufacturing and dicing of semiconductor wafer manufacturing have similar production environment.

There are balanced, setup times dominant and processing time dominant datasets in this Scheduling Research 2005 [4] database. In this research we made the comparisons using balanced datasets. Running the algorithm using the remaining datasets might give interesting results. In this research, we are applying local search in each generation. Determining the conditions of applying local search may accelerate the algorithm. In order get closer to the real system, machine eligibility would be included into the algorithms. Generating hyperheuristics through the use of different metaheuristics’ strengths may be a worthwhile avenue of research. Application of the proposed algorithm to multiobjective fitness functions would be a novel research topic. Other constraints, such as priority conditions, could be incorporated into future research. The other parallel machines problems like batch-processing machines seem also interesting.

Appendix

Here we give pseudocode for the local search. For brevity only the key steps are detailed.

C# notation is used.

See Algorithm 1.
For all generations $G$
For the population $P$
For all chromosomes $C$
$i = 0$ to $g$
$j = 1$ to $g$
If $Gene(i)_{Machine} == Gene(j)_{Machine}$ → (inter-machine interchange)
If $j = i + 1$ → (adjacent jobs) (Figure 5)
$a = 0; b = 0; c = 0;$
If $i > 0$ & $Gene(i)_{Machine} == Gene(i - 1)_{Machine}$
$a = (ST_{Machine}(i)Job(i - 1), j) - ST_{Machine}(i)Job(i - 1, j) + (ST_{Machine}(i)Job(i - 1), j) - ST_{Machine}(i)Job(i - 1, j))$ → (“a” on Figure 5)
If $i == 0$ || $Gene(i)_{Machine} != Gene(i - 1)_{Machine}$
$b = (ST_{Machine}(i)Job(i, j) - ST_{Machine}(i)Job(i, j) + (ST_{Machine}(i)Job(i, j) - ST_{Machine}(i)Job(i, j))$ → (“b” on Figure 5)
If $j < g - 1$ & $Gene(j)_{Machine} == Gene(j + 1)_{Machine}$
$c = ST_{Machine}(j)Job(j, j + 1) - ST_{Machine}(j)Job(j, j + 1) → (“c” on Figure 5)
If $a + b + c < 0$
Let to interchange $Gene(i)_{job}$ and $Gene(j)_{job}$ on the schedule
Calibrate random key numbers → (Section 4.2.2, item “i”)
If $j > i + 1$ → (non-adjacent jobs) (Figure 6)
$a = 0; b = 0; c = 0;$
If $i > 0$ & $Gene(i)_{Machine} == Gene(i - 1)_{Machine}$
$a = (ST_{Machine}(i)Job(i - 1, j) - ST_{Machine}(i)Job(i - 1, j) + (ST_{Machine}(i)Job(i - 1, j) - ST_{Machine}(i)Job(i - 1, j))))$
If $i == 0$ || $Gene(i)_{Machine} != Gene(i - 1)_{Machine}$
$b = (ST_{Machine}(i)Job(i, j) - ST_{Machine}(i)Job(i, j) + (ST_{Machine}(i)Job(i, j) - ST_{Machine}(i)Job(i, j))$ → (“b” on Figure 5)
If $j < g - 1$ & $Gene(j)_{Machine} == Gene(j + 1)_{Machine}$
$c = ST_{Machine}(j)Job(j, j + 1) - ST_{Machine}(j)Job(j, j + 1) → (“c” on Figure 5)
If $a + b + c < 0$
Let to interchange $Gene(i)_{job}$ and $Gene(j)_{job}$ on the schedule
Calibrate random key numbers → (Section 4.2.2, item “i”)
If $Gene(i)_{Machine} != Gene(j)_{Machine}$ → (intra-machine exchanging) (Figure 7)
$a = 0; a_1 = 0; a_2 = 0; b = 0; b_1 = 0; b_2 = 0; MakeSpan_{D_1} = 0; MakeSpan_{D_2} = 0;$
$a = CT_{Gene(i)_{Machine}} + PT_{Machine(i)Job(j)} - PT_{Machine(i)Job(j)}$
If $i > 0$ & $Gene(i)_{Machine} == Gene(i - 1)_{Machine}$
$a_1 = (ST_{Machine}(i)Job(i - 1, j) - ST_{Machine}(i)Job(i - 1, j))$
If $i == 0$ || $Gene(i)_{Machine} != Gene(i - 1)_{Machine}$
$a_1 = (ST_{Machine}(i)Job(i, j) - ST_{Machine}(i)Job(i, j)) → (Gene(i)_{job} is the first job on Gene(i)_{Machine})$
If $Gene(i)_{Machine} == Gene(i + 1)_{Machine}$
$b_1 = (ST_{Machine}(i)Job(i, j) - ST_{Machine}(i)Job(i, j)) → (Gene(i)_{job} is the first job on Gene(i)_{Machine})$
If $j < g - 1 & Gene(j)_{Machine} == Gene(j + 1)_{Machine}$
$b_2 = (ST_{Machine}(j)Job(j, j + 1) - ST_{Machine}(j)Job(j, j + 1)) → (Gene(j)_{job} is the first job on Gene(j)_{Machine})$
If $j < g - 1 & Gene(j)_{Machine} == Gene(j + 1)_{Machine}$
$b_3 = (ST_{Machine}(j)Job(j, j + 1) - ST_{Machine}(j)Job(j, j + 1)) → (Gene(j)_{job} is the first job on Gene(j)_{Machine})$
If $j < g - 1 & Gene(j)_{Machine} == Gene(j + 1)_{Machine}$
$b_3 = (ST_{Machine}(j)Job(j, j + 1) - ST_{Machine}(j)Job(j, j + 1)) → (Gene(j)_{job} is the first job on Gene(j)_{Machine})$
$a = a + a_1 + a_2$
$b = b + b_1 + b_3$
MakeSpan_{D_1} = Max(CT_{Gene(i)_{Machine}}; CT_{Gene(j)_{Machine}})
MakeSpan_{D_2} = Max(a; b)
If MakeSpan_{D_2} < MakeSpan_{D_1}
Let to exchange $Gene(i)_{job}$ and $Gene(j)_{job}$ on the schedule
Calibrate random key numbers → (Section 4.2.2, item “ii”)

Algorithm 1: Local search.
Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] E. Vallada and R. Ruiz, “A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times,” European Journal of Operational Research, vol. 211, no. 3, pp. 612–622, 2011.

[2] A. H. Kashan, B. Karimi, and F. Jolai, “An effective hybrid multi-objective genetic algorithm for bi-criteria scheduling on a single batch processing machine with non-identical job sizes,” Engineering Applications of Artificial Intelligence, vol. 23, no. 6, pp. 911–922, 2010.

[3] F. Yalaoui and C. Chu, “An efficient heuristic approach for parallel machine scheduling with job splitting and sequence-dependent setup times,” IEEE Transactions, vol. 35, no. 2, pp. 183–190, 2003.

[4] Scheduling Research. 2005. http://schedulingresearch.com/.

[5] A. Allahverdi, C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov, “A survey of scheduling problems with setup times or costs,” European Journal of Operational Research, vol. 187, no. 3, pp. 985–1032, 2008.

[6] X. Zhu and W. E. Wilhelm, “Scheduling and lot sizing with sequence-dependent setup: a literature review,” IEEE Transactions, vol. 38, no. II, pp. 987–IOOT, 2006.

[7] K. Li and S. L. Yang, “Non-identical parallel-machine scheduling research with minimizing total weighted completion times: models, relaxations and algorithms,” Applied Mathematical Modelling, vol. 33, no. 4, pp. 2145–2158, 2009.

[8] Y. K. Lin, M. E. Pfund, and J. W. Fowler, “Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems,” Computers and Operations Research, vol. 38, no. 6, pp. 901–916, 2011.

[9] Y. Park, S. Kim, and Y. H. Lee, “Scheduling jobs on parallel machines applying neural network and heuristic rules,” Computers and Industrial Engineering, vol. 38, no. 1, pp. 189–202, 2000.

[10] Y. H. Lee, K. Bhaskaran, and M. Pinedo, “A heuristic to minimize the total weighted tardiness with sequence-dependent setups,” IEEE Transactions, vol. 29, no. 1, pp. 45–52, 1997.

[11] N. van Hop and N. N. Nagarur, “The scheduling problem of PCBs for multiple non-identical parallel machines,” European Journal of Operational Research, vol. 158, no. 3, pp. 577–594, 2004.

[12] M. Zandieh, S. M. T. F. Ghomi, and S. M. M. Huseini, “An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times,” Applied Mathematics and Computation, vol. 180, no. 1, pp. 111–127, 2006.

[13] J. B. Wang, “Single machine scheduling with common due date and controllable processing times,” Applied Mathematics and Computation, vol. 174, no. 2, pp. 1245–1254, 2006.

[14] J. Behnamian, M. Zandieh, and S. M. T. Fatemi Ghomi, “Parallel-machine scheduling problems with sequence-dependent setup times using an ACO, SA and VNS hybrid algorithm,” Expert Systems with Applications, vol. 36, no. 6, pp. 9637–9644, 2009.

[15] T. Yang, “An evolutionary simulation-optimization approach in solving parallel-machine scheduling problems—a case study,” Computers and Industrial Engineering, vol. 56, no. 3, pp. 1126–1136, 2009.

[16] S. Balin, “Non-identical parallel machine scheduling using genetic algorithm,” Expert Systems with Applications, vol. 38, no. 6, pp. 6814–6821, 2011.

[17] T. Keskinurk, M. B. Yildirim, and M. Barut, “An ant colony optimization algorithm for load balancing in parallel machines with sequence-dependent setup times,” Computers and Operations Research, vol. 39, no. 6, pp. 1225–1235, 2012.

[18] G. Rabadi, R. J. Moraga, and A. Al-Salem, “Heuristics for the unrelated parallel machine scheduling problem with setup times,” Journal of Intelligent Manufacturing, vol. 17, no. 1, pp. 85–97, 2006.

[19] J. P. Arnaout, G. Rabadi, and R. Musa, “A two-stage Ant Colony Optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times,” Journal of Intelligent Manufacturing, vol. 21, no. 6, pp. 693–701, 2010.

[20] P. C. Chang and S. H. Chen, “Integrating dominance properties with genetic algorithms for parallel machine scheduling problems with setup times,” Applied Soft Computing Journal, vol. 11, no. 1, pp. 1263–1274, 2011.

[21] D. Yilmaz Eroglu, H. C. Oztutlu, and S. Oztutlu, “Genetic algorithm with local search for the unrelated parallel machine scheduling problem with sequence-dependent setup times,” International Journal of Production Research. In press.

[22] D. W. Kim, K. H. Kim, W. Jang, and F. Frank Chen, “Unrelated parallel machine scheduling with setup times using simulated annealing,” Robotics and Computer-Integrated Manufacturing, vol. 18, no. 3-4, pp. 223–231, 2002.

[23] Y. D. Kim, S. O. Shim, S. B. Kim, Y. C. Choi, and H. M. Yoon, “Parallel machine scheduling considering a job-splitting property,” International Journal of Production Research, vol. 42, no. 21, pp. 4531–4546, 2004.

[24] S. O. Shim and Y. D. Kim, “A branch and bound algorithm for an identical parallel machine scheduling problem with a job splitting property,” Computers and Operations Research, vol. 35, no. 3, pp. 863–875, 2008.

[25] W. Xing and J. Zhang, “Parallel machine scheduling with splitting jobs,” Discrete Applied Mathematics, vol. 103, no. 1–3, pp. 259–269, 2000.

[26] D. N. Tahar, F. Yalaoui, C. Chu, and L. Amodeo, “A linear programming approach for identical parallel machine scheduling with job splitting and sequence-dependent setup times,” International Journal of Production Economics, vol. 99, no. 1-2, pp. 63–73, 2006.

[27] P. Serafini, “Scheduling jobs on several machines with the job splitting property,” Operations Research, vol. 44, no. 4, pp. 617–628, 1996.

[28] D. Yilmaz Eroglu, H. C. Ozmutlu, and S. A. Koksal, “A genetic algorithm for the unrelated parallel machine scheduling problem with job splitting and sequence-dependent setup times—Loom scheduling,” Tekstil ve Konfeksiyon. In press.

[29] C. Pimentel, F. P. E. Alvelos, A. Duarte, and J. M. V. de Carvalho, “Exact and heuristic approaches for lot splitting and scheduling on identical parallel machines,” International Journal of Manufacturing Technology and Management, vol. 22, no. 1, pp. 39–57, 2011.
[30] J. A. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan, Ann Arbor, Mich, USA, 1975.

[31] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, Mass, USA, 1989.

[32] R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms*, John Wiley & Sons, New Jersey, NJ, USA, 2nd edition, 2004.