Genetic algorithms for hybrid job-shop problems with minimizing the makespan and mean flow time

Omid Gholami
Islamic Azad University - Mahmudabad center, Nour Branch, Mahmudabad, Iran.
email: gholami@iaumah.ac.ir

Yuri N. Sotskov
United Institute of Informatics Problems, National Academy of Sciences of Belarus, Surganova Str 6, Minsk 220012, Belarus.
email: sotskov@newman.bas-net.by; sotskov48@mail.ru

Frank Werner
Faculty of Mathematics, Otto-von-Guericke-University, Magdeburg, Germany.
email:frank.werner@ovgu.de

Abstract: We address a generalization of the classical job-shop problem which is called a hybrid job-shop problem. The criteria under consideration are the minimization of the makespan and mean flow time. In the hybrid job-shop, machines of type k are available for processing the specific subset $\mathcal{O}^k$ of the given operations. Each set $\mathcal{O}^k$ may be partitioned into subsets for their processing on the machines of type k. Solving the hybrid job-shop problem implies the solution of two subproblems: an assignment of all operations from the set $\mathcal{O}^k$ to the machines of type k and finding optimal sequences of the operations for their processing on each machine. In this paper, a genetic algorithm is developed to solve these two subproblems simultaneously. For solving the subproblems, a special chromosome is used in the genetic algorithm based on a mixed graph model. Computational results for the benchmark instances show that the proposed genetic algorithm is rather efficient for both criteria.

Keywords: Scheduling, Hybrid job-shop, Makespan, Mean flow time; Genetic algorithm

MSC classification: 90 B 35

May 2, 2016

1 Introduction

In the classical job-shop scheduling problem denoted as $J|r_i|\Phi$ in [21], each job from the set $\mathcal{J}$ has to be processed on each of m different machines from the set $\mathcal{M}$. A job $J_i \in \mathcal{J}$ is ready for processing from the release time $r_i \geq 0$, and the processing of the job J_i consists of m linearly ordered operations. Each operation should be processed on a specific machine from the set $\mathcal{M}$. The problems $J|r_i|C_{max}$ and $J|r_i|\sum C_i$ arise in many applications and are strongly NP-hard [8, 15, 21]. A lot of exact and heuristic algorithms have been developed for solving the problem $J|r_i|C_{max}$ and the problem $J|r_i|\sum C_i$ [1, 2, 6, 7, 11, 14, 16, 20, 22, 25, 27, 32, 37].

In industries, to have a better throughput, increasing the production performance, or because of fault tolerance more than one machine of the same type may be available. In such cases, a flexible

job-shop problem (denoted as $FJ|r_i|\Phi$) or a hybrid job-shop problem $HJ|r_i|\Phi$ have to be solved. The difference between the problems $HJ|r_i|\Phi$ and $FJ|r_i|\Phi$ is that all the parallel machines of the same type in the problem $HJ|r_i|\Phi$ are identical (i.e., they need the same time to process an operation). In the problem $FJ|r_i|\Phi$, the parallel machines may have different speeds and (or) an operation may be processed on several parallel machines.

Both problems $FJ|r_i|\Phi$ and $HJ|r_i|\Phi$ are more complicated than the job-shop problem $J|r_i|\Phi$ since the problems $FJ|r_i|\Phi$ and $HJ|r_i|\Phi$ contain the subproblem $P|r_i|\Phi$ for an optimal assignment of the operations to parallel machines. The latter problem $P|r_i|\Phi$ even with the simple criterion $\Phi = C_{max}$ is binary NP-hard [21] if there are only two identical machines and the release times are equal, $r_i = r$, for all jobs $J_i \in \mathcal{J}$. Therefore, in the problem $HJ|r_i|C_{max}$, the NP-hardness of the problem $J|r_i|C_{max}$ is combined with the NP-hardness of the problem $P|r_i|C_{max}$. Because of the high complexity, researchers and practitioners are forced to use heuristic algorithms for solving the problems $FJ|r_i|\Phi$ and $HJ|r_i|\Phi$ with large sizes. Next, we survey such scheduling algorithms and the results obtained.

A straightforward approach to solve the problem $FJ||\Phi$ is the decomposition of the original problem into a sequence of subproblems [33], which are job-shop scheduling problems $J||C_{max}$. The assignment subproblem was solved using priority dispatching rules and then the resulting job-shop problem $J||C_{max}$ was solved using a heuristic algorithm. Ferrer et al. [12] worked on the effects of the priority dispatching rules on the scheduling performance. A grammar based flexible job-shop algorithm has been developed. The performance of the algorithm was analyzed using computational experiments.

Wang and Yu [38] solved a production scheduling problem with machine availability constraints via a preventive maintenance. They considered a scheduling problem with both fixed and non-fixed machine availabilities. A heuristic algorithm based on a filtered beam search has been developed to solve the problem $FJ|r_i|\Phi$. A branching scheme was provided to incorporate the machine availability constraints and the maintenance resource ones.

Gao et al. [13] worked on the problem $FJ|r_i|\Phi$ with three objectives: minimizing the makespan, minimizing the maximal machine workload, and minimizing the total workload. They used a genetic algorithm with a variable neighborhood descent. In order to strengthen the search ability, the genetic algorithm was improved by a variable neighborhood descent that involves a local search of moving one or two operations.

Other researches (see [3, 10, 23, 34, 40]) developed genetic algorithms combined with different search techniques. Rossi and Boschi [34] used a genetic algorithm and an ant colony optimization to solve the problem $HJ|r_i|\Phi$. Al-Hinai and ElMekkawy [3] worked on the problem of finding robust and stable solutions for the problem $FJ|r_i|\Phi$ with random machine breakdowns. Chiang and Lin [10] proposed a multi-objective evolutionary algorithm to solve the problem $HJ|r_i|\Phi$ in a Pareto manner aimed to seek for the non-dominated schedules. Xing et al. [40] developed a simulation model to solve the multi-objective problem $FJ|r_i|\Phi$. Hmidaa et al. [23] presented neighborhood structures related to the assignment problem and the sequencing one. They proposed a climbing discrepancy search for solving the problem $FJ|r_i|\Phi$.

The surveys [17, 39] and the above papers show that evolutionary techniques like a genetic algorithm are mainly used for solving the problem $FJ|r_i|\Phi$. Using a fast computer, it is now possible to run the same evolutionary algorithm many times in order to increase the quality of the schedule obtained for the problem $FJ|r_i|\Phi$ [5, 13, 28].

A special case of the flexible job-shop problem is the problem $HJ|r_i|\Phi$, where all machines available to process an operation are identical. The problem $HJ|r_i|C_{max}$ is encountered in a flexible manufacturing system [17], train timetabling [24, 30], and in many other fields. Next, we survey algorithms developed for the problems $HJ|r_i|C_{max}$ and $HJ|r_i|\sum C_i$.

Liu and Kozan [29] used a hybrid job-shop problem to solve the train timetabling problem. The train timetabling was modeled as a hybrid job-shop problem such that the trains, single-track sections and multiple-track sections were synonymous with the jobs, single machines and parallel machines. A train movement on railway sections was considered as the operations in the job-shop. Liu and Kozan [29] considered blocking or hold-while-wait constraints to simulate a fixed buffer size for the trains, which means that a track section cannot be released and must hold the train until the next railway section on the routing becomes free. First, the problem was solved by a shifting bottleneck algorithm [2] without considering blocking conditions. Then a feasibility satisfaction procedure was used to solve and analyze the blocking factor via an alternative graph modeling.

The algorithm developed in [9] consists of a genetic algorithm for scheduling the operations and for the machine selection. The minimization of the total tardiness, the total machine idle time and the makespan have been considered in [9]. Rossi and Boschi [34] reported their research on the hybrid job-shop problem, where a genetic algorithm was combined with an ant colony optimization. A modular approach was adopted to obtain an easy scalable parallel evolutionary ant colony framework.

Three algorithms have been developed in [36] for the problems $HJ|r_i|\Phi$, where the criterion Φ was the minimization of the makespan or the sum of job completion times. The developed algorithms include the sequencing and assigning stages. In the sequencing stage, the job-shop problem $J|r_i|\Phi$ was solved. The problem $J|r_i|\Phi$ was modelled by a mixed graph G. In order to resolve a conflict arising between two operations processed on the same machine, the algorithm should substitute a conflict edge in the mixed graph G by an arc incident to the same vertexes.

In [18, 19], an adaptive algorithm and a neural network algorithm have been developed for solving the problem $HJ|r_i|\Phi$. These algorithms have a learning stage for solving the problems $HJ|r_i|\Phi$ with a small size optimally. Then the knowledge about the optimal scheduling of the conflict operations was used for solving the problem $HJ|r_i|\Phi$ with a large size. Ong and Maulana [31] developed an algorithm to solve the non-delay job-shop problem with parallel machines. The developed algorithm was based on the shifting bottleneck one [2] and tried to reduce the makespan value using priority dispatching rules.

In this paper, a genetic algorithm is developed for solving both problems $HJ|r_i|C_{max}$ and $HJ|r_i|\sum C_i$. In Sections 2 and 3, the problem setting and the mixed graph model are described. In Section 4, a genetic algorithm for the problem $HJ|r_i|\Phi$ is developed. In Section 5.1, a mixed graph for presenting the problem $HJ|r_i|\Phi$ is described. An upgraded genetic algorithm for solving the problem $HJ|r_i|\Phi$ is discussed in Section 5.2. Computational results for the developed genetic algorithms are presented in Section 6.

2 Settings for the problems $J|r_i|\Phi$ and $HJ|r_i|\Phi$

There are n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ that have to be processed on m machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. The set $\mathcal{M}$ is partitioned into $\varphi \geq 2$ subsets as follows:

$$\mathcal{M} = \mathcal{M}_1 \bigcup \mathcal{M}_2 \bigcup \ldots \bigcup \mathcal{M}_\varphi, \mathcal{M}_k \bigcap \mathcal{M}_u = \emptyset,\ k \neq u,$$

where $\mathcal{M}_k \neq \emptyset,\ k \in \{1, 2, \ldots, \varphi\}$, is the subset of the identical machines of the type k.

In the job-shop problem $J|r_i|\Phi$, each set $\mathcal{M}_k$ consists of a single machine: $\mathcal{M}_k = \{M_k\}$ for each type $k \in \{1, 2, \ldots, \varphi\}$ of machines, where $\varphi = m$.

In the hybrid job-shop problem $HJ|r_i|\Phi$, each set $\mathcal{M}_k$ consists of φ_k identical machines of the type $k \in \{1, 2, \ldots, \varphi\}$, where inequality $\varphi_u \geq 2$ holds for at least one machine type $u \in \{1, 2, \ldots, \varphi\}$. Let the machines from the set $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ be indexed as follows:

$$\mathcal{M}_k = \left\{ M_{\sum_{u=0}^{k-1} \varphi_u + 1}, M_{\sum_{u=0}^{k-1} \varphi_u + 2}, \ldots, M_{\sum_{u=0}^{k} \varphi_u} \right\},\ k \in \{1, 2, \ldots, \varphi\},$$

where $\varphi_0 = 0$ and $\sum_{u=1}^{\varphi} \varphi_u = m$.

The processing of a job $J_i \in \mathcal{J}$ consists of a set $\mathcal{O}_i$ of φ ordered operations:

$$\mathcal{O}_i = \left\{ Q_{i1}^{\mu(i1)}, Q_{i2}^{\mu(i2)}, \ldots, Q_{i\varphi}^{\mu(i\varphi)} \right\},$$

where operation $Q_{ij}^{\mu(ij)}$ denotes the processing of the job $J_i \in \mathcal{J}$ at the stage $j \in \{1, 2, \ldots, \varphi\}$ on a machine of the type $\mu(ij) \in \{1, 2, \ldots, \varphi\}$ during the processing time $p_{ij} > 0$.

Preemptions of the operations are not allowed, i.e., if operation $Q_{ij}^{\mu(ij)} \in \mathcal{O}_i$ is started at time $s(Q_{ij}^{\mu(ij)})$, this operation must be completed at time $c(Q_{ij}^{\mu(ij)}) = s(Q_{ij}^{\mu(ij)}) + p_{ij}$. The first operation $Q_{i,1}^{\mu(i,1)}$ of the job $J_i \in \mathcal{J}$ is ready for processing from time $r_i \geq 0$, called the release time of the job J_i. In the problem $HJ|r_i|\Phi$, each operation $Q_{ij}^{\mu(ij)}$ may be processed on any machine of the type $\mu(ij)$. No machine $M_r \in \mathcal{M}$ can process more than one operation at a time. The operation $Q_{ij}^{\mu(ij)}$ cannot be started until the previous operation $Q_{i,j-1}^{\mu(i,j-1)}$ of the job J_i is completed, i.e., the inequality

$$c(Q_{i,j-1}^{\mu(i,j-1)}) \leq s(Q_{ij}^{\mu(ij)})$$

must hold for each job $J_i \in \mathcal{J}$ and stage $j \in \{2, 3, \ldots, m\}$. The completion time C_i of the job $J_i \in \mathcal{J}$ is equal to the completion time of the last operation $Q_{i\varphi}^{\mu(i\varphi)}$ of this job:

$$C_i = c(Q_{i\varphi}^{\mu(i\varphi)}).$$

The problem $HJ|r_i|C_{max}$ is to find an assignment of the operations from the set

$$\mathcal{O}^k = \bigcup_{\mu(ij)=k} Q_{ij}^{\mu(ij)}$$

to the machines from the set $\mathcal{M}_k$, where $k \in \{1, 2, \ldots, \varphi\}$, and to define the completion times $c(Q_{ij}^{\mu(ij)})$ for all operations

$$Q_{ij}^{\mu(ij)} \in \mathcal{O}^* = \bigcup_{k=1}^{\varphi} \mathcal{O}^k$$

such that the resulting schedule

$$S = \left(c(Q_{1,1}^{\mu(1,1)}), c(Q_{1,2}^{\mu(1,2)}), \ldots, c(Q_{1,\varphi}^{\mu(1,\varphi)}), \ldots, c(Q_{n,1}^{\mu(n,1)}), c(Q_{n,2}^{\mu(n,2)}), \ldots, c(Q_{n,\varphi}^{\mu(n,\varphi)})\right) \qquad (1)$$

has a minimal makespan value $C_{max}(S) = \max\{C_i(S) \mid J_i \in \mathcal{J}\}$.

The problem $HJ|r_i|\sum C_i$ is to find the assignments of the operations $\mathcal{O}^k$ to the machines from the set $\mathcal{M}_k$ for each $k \in \{1, 2, \ldots, \varphi\}$, and to define the completion times $c(Q_{ij}^{\mu(ij)})$ for all operations $Q_{ij}^{\mu(ij)} \in \mathcal{O}^*$ such that the resulting schedule S defined in (1) has a minimal sum $\sum C_i(S) = \sum_{i=1}^{n} C_i(S)$ of the job completion times.

Since for the job-shop problem $J|r_i|\Phi$, there exists a single machine of each type, i.e., $|\mathcal{M}_1| = |\mathcal{M}_2| = \ldots = |\mathcal{M}_\varphi| = 1$ with $\varphi = m$, there is no need to seek assignments of the operations $\mathcal{O}^k$ to the machines $\mathcal{M}_k$ (the assignments of all the operations $\mathcal{O}^*$ to the machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$, with $m = \varphi$, are given as input data). In this case, the problem $HJ|r_i|\Phi$ is transformed into the problem $J|r_i|\Phi$, which is considered in the next two sections.

3 A mixed graph model for the job-shop problem $J|r_i|\Phi$

The problem $J|r_i|\Phi$ can be modeled using a mixed (or disjunctive) graph $G = (\mathcal{O}, A, E)$ with the set $\mathcal{O}$ of vertexes, the set A of arcs, and the set E of edges [6, 37]. The set $\mathcal{O}^*$ is the set of operations $Q_{ij}^{\mu(ij)}$, $J_i \in \mathcal{J}$, $j \in \{1, 2, \ldots, m\}$, the weight $p_{ij} > 0$ being prescribed to the vertex $Q_{ij}^{\mu(ij)} \in \mathcal{O}^*$. For each job $J_i \in \mathcal{J}$, the vertex Q_i is added to the set $\mathcal{O}$ of the vertexes in the mixed graph G, the weight $r_i \geq 0$ being prescribed to the vertex Q_i with $i \in \{1, 2, \ldots, n\}$, where

$$\mathcal{O} = \bigcup_{i=1}^{n} \{Q_i\} \bigcup \mathcal{O}^*.$$

The consecutive operations $Q_{i,j-1}^{\mu(i,j-1)}$ and $Q_{ij}^{\mu(ij)}$ of the job $J_i \in \mathcal{J}$ are connected by the arc $(Q_{i,j-1}^{\mu(i,j-1)}, Q_{ij}^{\mu(ij)}) \in A$, $2 \leq j \leq m$, in the mixed graph $G = (\mathcal{O}, A, E)$. The arc $(Q_{i,j-1}^{\mu(i,j-1)}, Q_{ij}^{\mu(ij)})$ expresses that the operation $Q_{ij}^{\mu(ij)}$ can be started only after the time when the previous operation $Q_{i,j-1}^{\mu(i,j-1)}$ of the job J_i is completed (*a precedence constraint*). Along with the above arcs, the set A includes the following subset of arcs:

$$\bigcup_{i=1}^{n} (Q_i, Q_{i1}^{\mu(i1)}).$$

Each such arc $(Q_i, Q_{i1}^{\mu(i1)})$ expresses that the processing of the job $J_i \in \mathcal{J}$ cannot be started before the release time $r_i \geq 0$ of the job J_i.

Any two operations $Q_{ij}^{\mu(ij)} \in \mathcal{O}^*$ and $Q_{lk}^{\mu(lk)} \in \mathcal{O}^*$ that have to be processed on the machine $M_h \in \mathcal{M}$, where $\mu(ij) = \mu(lk) = h$, cannot be simultaneously processed on the same machine. In the mixed graph $G = (\mathcal{O}, A, E)$, this *resource constraint* is expressed by the edge $[Q_{ij}^{\mu(ij)}, Q_{lk}^{\mu(lk)}] \in E$. Thus, the above mixed graph $G = (\mathcal{O}, A, E)$ determines both resource and precedence constraints for the job-shop problem $J|r_i|\Phi$. Since an operation preemption is not allowed, the schedule S defined in

(1), which is feasible for the mixed graph $G = (\mathcal{O},\, A, E)$, may be determined as the ordered set

$$S = \Big(c(Q_{1,1}^{\mu(1,1)}), c(Q_{1,2}^{\mu(1,2)}), \ldots, c(Q_{1m}^{\mu(1m)}), \ldots, c(Q_{n,1}^{\mu(n,1)}), c(Q_{n,2}^{\mu(n,2)}), \ldots, c(Q_{nm}^{\mu(nm)})\Big)$$

of the completion times of the operations $Q_{ij}^{\mu(ij)} \in \mathcal{O}^*$ such that the precedence constraint

$$s(Q_{lk}^{\mu(lk)}) - s(Q_{ij}^{\mu(ij)}) \geq p_{ij} \tag{2}$$

is satisfied for each arc $(Q_{ij}^{\mu(ij)}, Q_{lk}^{\mu(lk)}) \in A$ and the resource constraint

$$\text{either} \quad s(Q_{lk}^{\mu(lk)}) - s(Q_{ij}^{\mu(ij)}) \geq p_{ij} \quad \text{or} \quad s(Q(_{ij}^{\mu(ij)}) - s(Q_{lk}^{\mu(lk)}) \geq p_{lk} \tag{3}$$

is satisfied for each edge $[Q_{ij}^{\mu(ij)}, Q_{lk}^{\mu(lk)}] \in E$. Using the mixed graph $G = (\mathcal{O},\, A, E)$ to construct a schedule $S(G_r)$ for the problem $J|r_i|\Phi$, one can replace each edge $[Q_{ij}^{\mu(ij)}, q_{lk}^{\mu(lk)}] \in E$ either by the arc $(Q_{ij}^{\mu(ij)}, Q_{lk}^{\mu(lk)})$ or the arc $(Q_{lk}^{\mu(lk)}, Q_{ij}^{\mu(ij)})$ respecting the resource constraint (3) in a way such that no a circuit arises in the resulting digraph $G_r = (\mathcal{O},\, A \cup A_r, \varnothing)$. As a result, the mixed graph $G = (\mathcal{O},\, A, E)$ will be transformed into the circuit-free digraph $G_r = (\mathcal{O},\, A \bigcup A_r, \varnothing)$, in which both precedence constraints (2) and resource constraints (3) are taken into account by the arc set $A \bigcup A_r$. Let $\Gamma(G)$ denote the set of all digraphs $G_r = (\mathcal{O},\, A \cup A_r, \varnothing)$, where each edge $[Q_{ij}^{\mu(ij)}, Q_{uv}^{\mu(uv)}] \in E$ is replaced either by the arc $(Q_{ij}^{\mu(ij)}, Q_{uv}^{\mu(uv)}) \in A_r$ or the arc $(Q_{uv}^{\mu(uv)}, Q_{ij}^{\mu(ij)}) \in A_r$. A schedule S is called semi-active if no operation can start earlier without delaying the processing of some other operation from the set $\mathcal{O}^*$ or without altering the processing sequence of the operations on any machines from the set $\mathcal{M}$. The mixed graph approach for solving the job-shop problem $J|r_i|\Phi$ is based on the following claim [37].

Lemma 1 *The circuit-free digraph $G_r = (\mathcal{O},\, A \cup A_r, \varnothing) \in \Gamma(G)$ determines a semi-active schedule $S(G_r)$ for the problem $J|r_i|\Phi$ and vice-versa.*

Let $\Gamma^*(G)$ denote the set of all circuit-free digraphs $G_r = (\mathcal{O},\, A \cup A_r, \varnothing)$ belonging to the set $\Gamma(G)$. Due to Lemma 1, there exists a one-to-one correspondence between the set of semi-active schedules $S(G_r)$ existing for the problem $J|r_i|\Phi$ and the set $\Gamma^*(G) \subseteq \Gamma(G)$ of the circuit-free digraphs $G_r = (\mathcal{O},\, A \cup A_r, \varnothing)$ generated by the mixed graph G via orienting the edges E.

For the problem $J|r_i|C_{max}$, the objective function value $C_{max}(S(G_r))$ is equal to the weight $p_{ij} + p_{uv} + \ldots + p_{ef}$ of a critical path $(Q_{ij}^{\mu(ij)}, Q_{uv}^{\mu(uv)}, \ldots, Q_{ef}^{\mu(ef)})$ existing in the digraph $G_r \in \Gamma^*(G)$. Thus, the following claim is correct.

Lemma 2 *A digraph $G_r \in \Gamma^*(G)$ is optimal for the problem $J|r_i|C_{max}$ if and only if a critical path in the digraph G_r has the minimal weight among all critical paths in the digraphs from the set $\Gamma^*(G)$.*

Similarly, the following claim is obtained for the problem $J|r_i|\sum C_i$.

Lemma 3 *The digraph $G_r \in \Gamma^*(G)$ is optimal for the problem $J|r_i|\sum C_i$ if and only if the objective function value $\sum C_i(S(G_r)) = \sum_{i=1}^{n} C_i(S(G_r))$ for the digraph G_r is minimal among the values $\sum C_i(S(G_k))$ for the digraphs $G_k \in \Gamma^*(G)$.*

The above claims may be generalized for any regular criterion Φ. (A criterion Φ is called *regular*, if the objective function $\Phi(C_1, C_2, \ldots, C_n)$ is non-decreasing for all arguments $C_1, C_2, \ldots, C_n$).

Lemma 4 *Let the criterion Φ be regular. Then the digraph $G_r \in \Gamma^*(G)$ is optimal for the problem $J|r_i|\Phi$ if and only if the objective function value*

$$\Phi(S(G_r)) = \Phi(C_1(S(G_r)), C_2(S(G_r)), \ldots, C_n(S(G_r)))$$

calculated for the digraph G_r is minimal among the values

$$\Phi(S(G_k)) = \Phi(C_1(S(G_k)), C_2(S(G_k)), \ldots, C_n(S(G_k)))$$

calculated for all the digraphs $G_k \in \Gamma^*(G)$.

We follow the monograph [37], where a schedule S is defined as left-continuous functions. In other words, for any job $J_i \in \mathcal{J}$, $j \in \{1, 2, \ldots, m\}$, the semi-interval $(s(Q_{ij}^k), c(Q_{ij}^k)]$ determines the time interval, in which the machine M_k processes the operation $Q_{ij}^k \in \mathcal{O}^*$ of the job J_i. Since any machine $M_k \in \mathcal{M}$ can process at most one job at a time, the semi-intervals $(s(Q_{ij}^k), c(Q_{ij}^k)]$ and $(s(Q_{uv}^k), c(Q_{uv}^k)]$ for the processing of operations Q_{ij}^k and Q_{uv}^k have no common points:

$$(s(Q_{ij}^k), c(Q_{ij}^k)] \bigcap (s(Q_{uv}^k), c(Q_{uv}^k)] = \emptyset.$$

4 A genetic algorithm for the problem $J|r_i|\Phi$

A genetic algorithm constructs a solution (an individual) for a problem by ordered *genes* known as a *chromosome*. To imitate the evolution in living nature, the chromosomes are combined or mutated to create a new individual in the constructed generation. While running a crossover operation, an offspring's chromosome is created by joining segments chosen alternately from each of the chromosomes of the two parents, which are of fixed length. By recombining portions of good individuals, this process is likely to create a better individual. For a mutation, a small portion of the current generation will be selected and transformations will occur in their genome. The transformation purpose is to maintain diversity within the constructed population and to inhibit premature convergence. A mutation induces a random walk through a search space of the problem. After several generations, new individuals will be created that probably will have a better objective function value.

As mentioned in Section 3 (Lemmas 1 and 4), the algorithm for the problem $J|r_i|\Phi$ must replace each edge $[Q_{ij}^{\mu(ij)}, Q_{uv}^{\mu(uv)}] \in E$ with $\mu(ij) = \mu(uv) = k$ either by the arc $(Q_{ij}^{\mu(ij)}, Q_{uv}^{\mu(uv)})$ (in this case, the operation $Q_{ij}^{\mu(ij)}$ must be completed before starting the operation $Q_{uv}^{\mu(uv)}$ on machine M_k) or by the symmetric arc $(Q_{uv}^{\mu(uv)}, Q_{ij}^{\mu(ij)})$ (in this case, the operation $Q_{uv}^{\mu(uv)}$ must be completed before starting the operation $Q_{ij}^{\mu(ij)}$ on machine M_k). Therefore, to solve the problem $J|r_i|\Phi$ by a genetic algorithm, the chromosome must be able to determine a sequence, in which all operations of the set $\mathcal{O}^k = \{Q_{ij}^{\mu(ij)} \mid J_i \in \mathcal{J}, \mu(ij) = k\}$ could be processed on a single machine M_k of the type k. For this purpose, a unique natural number (gene) q_u, $u \in \{1, 2, \ldots, nm-1, nm\}$, will be used to code each operation of the set $\mathcal{O}^* = \bigcup_{k=1}^{\varphi} \mathcal{O}^k$.

Let the ordered operations

$$\left(Q_{1,1}^{\mu(1,1)}, Q_{1,2}^{\mu(1,2)}, \ldots, Q_{1,m}^{\mu(1,m)}\right)$$

of the job $J_1 \in \mathcal{J}$ be coded by the following ordered numbers ($q_1 = 1, q_2 = 2, \ldots, q_m = m$), respectively. The ordered operations

$$\left(Q_{2,1}^{\mu(2,1)}, Q_{2,2}^{\mu(2,2)}, \ldots, Q_{2,m}^{\mu(2,m)}\right)$$

of the job $J_2 \in \mathcal{J}$ will be coded by the following ordered numbers ($q_{m+1} = m+1, q_{m+2} = m+2, \ldots, q_{2m} = 2m$), respectively. In general, the ordered operations

$$\left(Q_{i1}^{\mu(i1)}, Q_{i2}^{\mu(i2)}, \ldots, Q_{im}^{\mu(im)}\right)$$

of the job $J_i \in \mathcal{J}$ will be coded by the following ordered numbers

$$(q_{(i-1)m+1} = (i-1)m+1, q_{(i-1)m+2} = (i-1)m+2, \ldots, q_{im} = im),$$

respectively. The obtained numbers (genes) $g_u = u$, $u \in \{1, 2, \ldots, nm\}$, are ordered in the chromosome g^t, and determine the position for each operation $Q_{ij}^{\mu(ij)} \in \mathcal{O}^k$ with respect to the other operations $\mathcal{O}^k \setminus \{Q_{ij}^{\mu(ij)}\}$, which have to be processed on the machines M_k, $k \in \{1, 2, \ldots, m\}$. The chromosome g^t consists of nm ordered genes as follows:

$$g^t = (q_{k_1}, q_{k_2}, \ldots, q_{k_{nm-1}}, q_{k_{nm}}),$$

where $\{q_{k_1}, q_{k_2}, \ldots, q_{k_{nm}}\} = \{1, 2, \ldots, nm\}$.

Let us consider Example 1 of the job-shop problem $J|r_i|\Phi$ with four jobs, $\mathcal{J} = \{J_1, J_2, J_3, J_4\}$, and three machines, $\mathcal{M} = \{M_1, M_2, M_3\}$. The digraph $(\mathcal{O}, A, \emptyset)$ for this example is presented in Fig. 1 (for simplicity, the edges E of the mixed graph $G = (\mathcal{O}, A, E)$ are omitted in Fig. 1). Consider the second operation $Q_{ij}^{\mu(ij)} = Q_{1,2}^2$ of the job J_1 processed on machine M_2 at the stage $j = 2$. The upper index $\mu(ij) = \mu(1,2) = 2$ in the notation $Q_{1,2}^2 = Q_{ij}^{\mu(ij)}$ means that machine $M_2 \in \mathcal{M}$ has to process the operation $Q_{1,2}^2$. The other operations $Q_{ij}^{\mu(ij)}$ in the set $\mathcal{O}$ with the same index $\mu(ij) = 2$ (i.e., the operations $Q_{2,3}^2$, $Q_{3,1}^2$, and $Q_{4,3}^2$) are in conflict with operation $Q_{1,2}^2$, since they have to be processed on the same machine M_2. Since a machine may process only one operation at a time, and preemption of any operation is not allowed, to resolve these conflicts, it is necessary to determine a linear order (a sequence) of the operations $Q_{1,2}^2$, $Q_{2,3}^2$, $Q_{3,1}^2$, and $Q_{4,3}^2$ for their processing on the machine M_2.

The aim of solving this problem $J|r_i|\Phi$ is to determine the $m = 3$ linear orders (sequences) of the conflict operations for all machines $M_k \in \mathcal{M}$, $k \in \{1, 2, 3\}$, in such a way that the objective function value will be minimal (Lemma 4). Next, we show how a single chromosome g^t can determine all $m = 3$ sequences of the conflict operations $\mathcal{O}^k$, $k \in \{1, 2, 3\}$.

In the first step of the genetic algorithm, each operation $Q_{ij}^{\mu(ij)} \in \mathcal{O}$ has to be labeled with a natural number (gene) $q_u = u$ as it was explained (see Fig. 2). Initially, a chromosome g^t will be constructed from a random ordering of these genes that represents a heuristic solution of the job-shop problem $J|r_i|\Phi$. For example, the chromosome

$$g^t = (q_1, q_{10}, q_7, q_4, q_2, q_{11}, q_5, q_8, q_{12}, q_3, q_6, q_9) = (1, 10, 7, 4, 2, 11, 5, 8, 12, 3, 6, 9) \tag{4}$$

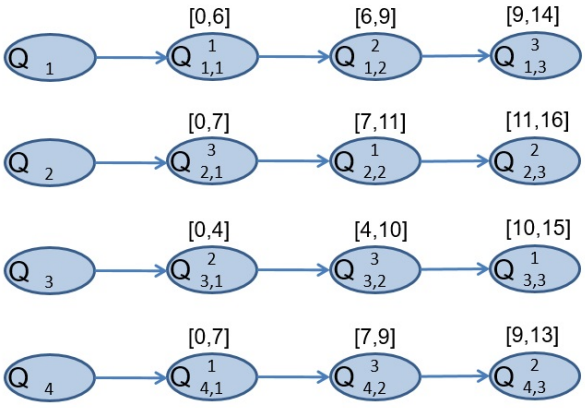

Figure 1: The digraph $(\mathcal{O}, A, \emptyset)$ for the job-shop problem with four jobs, $\mathcal{J} = \{J_1, J_2, J_3, J_4\}$, and three machines, $\mathcal{M} = \{M_1, M_2, M_3\}$.

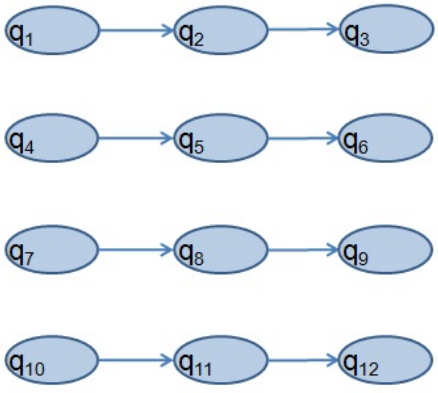

Figure 2: The vertexes of the set $\mathcal{O} = \{\bigcup_{k=1}^{\varphi} \mathcal{O}^k\}$ labeled by the natural numbers (genes) q_a, $a \in \{1, 2, \ldots, 12\}$.

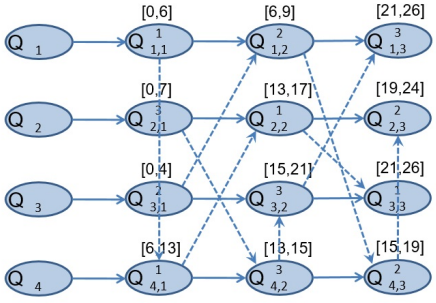

Figure 3: The digraph G_t (without transitive arcs) determined from the chromosome g^t.

determines a heuristic solution for the problem $J|r_i|\Phi$, whose input data are presented in Fig 1. Indeed, the positions, in which the numbers $q_a = a$ and $q_b = b$ coding the conflict operations $Q_{ij}^{\mu(ij)}$ and $Q_{uv}^{\mu(uv)}$ are placed in the chromosome g^t, determine the processing order of these operations $Q_{ij}^{\mu(ij)}$ and $Q_{uv}^{\mu(uv)}$ as follows. Let the number $q_a = a$ be located before the number $q_b = b$ (we denote this as $q_a \longrightarrow q_b$). This will mean that operation $Q_{ij}^{\mu(ij)}$ must be processed before operation $Q_{uv}^{\mu(uv)}$. Otherwise, if $q_b \longrightarrow q_a$, the operation $Q_{uv}^{\mu(uv)}$ must be processed before operation $Q_{ij}^{\mu(ij)}$. Therefore, the edge $[Q_{ij}^{\mu(ij)}, Q_{uv}^{\mu(uv)}]$ must be replaced either by the arc $(Q_{ij}^{\mu(ij)}, Q_{uv}^{\mu(uv)})$ in the mixed graph G or in the opposite case, by the symmetric arc $(Q_{uv}^{\mu(uv)}, Q_{ij}^{\mu(ij)})$. After the replacements of the edges by the arcs, the digraph $G_t = (\mathcal{O}, A \cup A_t, \varnothing) \in \Gamma(G)$ will be constructed with respect to the chromosome g^t.

In Fig. 1, the genes $q_1 = 1, q_5 = 5, q_9 = 9$ and $q_{10} = 10$ determine the operations, which are conflict ones for using machine M_1. In the chromosome g^t given in (4), the sequence of these genes is as follows: $q_1 \longrightarrow q_{10} \longrightarrow q_5 \longrightarrow q_9$. Therefore, the operation $Q_{1,1}^1$ will be processed first, then operation $Q_{4,1}^1$, then operation $Q_{2,2}^1$, and then operation $Q_{3,3}^1$. For machine M_2, the order of the corresponding genes is as follows: $q_7 \longrightarrow q_2 \longrightarrow q_{12} \longrightarrow q_6$. For machine M_3, the order of the corresponding genes is as follows: $q_4 \longrightarrow q_{11} \longrightarrow q_8 \longrightarrow q_3$. Figure 3 illustrates the digraph $G_t = (\mathcal{O}, A \cup A_t, \varnothing)$ determined from the chromosome g^t given in (4) for the problem $J|r_i|\Phi$, whose digraph $(\mathcal{O}, A, \emptyset)$ is presented in Fig. 1. Since the chromosome g^t was created randomly, the objective function value $\Phi(S(G_t))$ calculated for the digraph $G_t = (\mathcal{O}, A \cup A_t, \varnothing)$ could be considerably larger than that calculated for the optimal digraph constructed for the problem $J|r_i|\Phi$ under consideration.

5 A genetic algorithm for the hybrid job-shop problem $HJ|r_i|\Phi$

For a better job processing, one can use more than one machine of the type $k \in \{1, 2, \ldots, \varphi\}$. Such a machine flexibility opens a new circumstance in the solution space since each operation $Q_{ij}^{\mu(ij)} \in \mathcal{O}^k = \{Q_{ij}^{\mu(ij)} \,|\, J_i \in \mathcal{J}, \mu(ij) = k\}$, which has to be processed on a machine of the type $\mu(ij)$, has the freedom to be processed on any machine from the set $\mathcal{M}_k$. As a result, the operations $\mathcal{O}^k$ in the queue for processing on a machine of the type k may be partitioned into $|\mathcal{M}_k|$ subsets and operations of each such subset must be processed on the same machine M_u from the set $\mathcal{M}_k$. Thus, in solving the problem $HJ|r_i|\Phi$, the optimization problem of finding an optimal schedule for the job-shop problem $J|r_i|\Phi$ has to be combined with partitioning the operations $\mathcal{O}^k$ for each

machine type $k \in \{1, 2, \ldots, \varphi\}$.

Next, we show how the genetic algorithm presented in Section 4 may be generalized for sequencing and partitioning the operations $\mathcal{O}^k$ for each machine type $k \in \{1, 2, \ldots, \varphi\}$. To this end, a modification g^t_* of the above chromosome g^t will be used. The modified chromosome g^t_* will have the ability to partition the operations $\mathcal{O}^k$ and to represent a whole solution for the problem $HJ|r_i|\Phi$. In Section 5.1, a necessary and sufficient condition for partitioning the operations $\mathcal{O}^k$ in the digraph $G_t = (\mathcal{O}, A \cup A_t, \varnothing)$ is given. The modified chromosome g^t_* is introduced in Section 5.2.

5.1 A mixed graph model for the problem $HJ|r_i|\Phi$

A semi-active schedule $S(G_h)$ constructed for the problem $J|r_i|\Phi$ may be used as a heuristic solution for the problem $HJ|r_i|\Phi$, where exactly one machine $M_u \in \mathcal{M}_k$ processes all operations $\mathcal{O}^k$, $k \in \{1, 2, \ldots, \varphi\}$, i.e., the partition of the operations $\mathcal{O}^k$ into $|\mathcal{M}_k|$ subsets is as follows:

$$\mathcal{O}^k = \mathcal{O}^k \cup \emptyset \cup \emptyset \cup \ldots \cup \emptyset.$$

Thus, we obtain the following claim.

Lemma 5 *A circuit-free digraph $G_h = (\mathcal{O}, A \bigcup A_r, \varnothing) \in \Gamma(G)$ determines a semi-active schedule $S(G_h)$ for the problem $HJ|r_i|\Phi$ with the input data given by the mixed graph $G = (\mathcal{O}, A, E)$ and by the numbers $\varphi_k \geq 1$ of the machines of the types $k \in \{1, 2, \ldots, \varphi\}$.*

Obviously, the semi-active schedule $S(G_h)$ given in Lemma 5 may be very different from an optimal one for the problem $HJ|r_i|\Phi$ even if the schedule $S(G_h)$ is optimal for the corresponding problem $J|r_i|\Phi$. For decreasing the value $\Phi(S(G_h))$, one can use more machines from the set $\mathcal{M}_k$ to process the operations $\mathcal{O}^k$. Let the set E^* denote the subset of the set E and $\Gamma(G(E^*))$ denote the set of all digraphs $G_h(E^*) = (\mathcal{O}, A \cup A_h, \emptyset)$ generated by the mixed graph $G = (\mathcal{O}, A, E)$ via orienting all edges for the set E^*. The following theorem proven in [36] provides a criterion for the feasibility of the digraph $G_h(E^*) = (\mathcal{O}, A \cup A_h, \emptyset) \in \Gamma(G(E^*))$ for solving the problem $HJ|r_i|\Phi$.

Theorem 1 *The digraph $G_h(E^*) = (\mathcal{O}, A \cup A_h, \emptyset) \in \Gamma(G(E^*))$ with $E^* \subset E$ determines a semi-active schedule $S(G_h(E^*))$ for the problem $HJ|r_i|\Phi$ if and only if*

(a) the digraph $G_h(E^*)$ has no circuits and

(b) for any machine type $k \in \{1, 2, \ldots, \varphi\}$, at most φ_k operations from the set Q^k may have a common processing time interval $(t', t'']$, $t' < t''$.

A mixed graph approach for solving the problem $HJ|r_i|\Phi$ is based on Theorem 1 and Lemma 4. The corresponding algorithm must choose a digraph $G_h(E^*) = (\mathcal{O}, A \cup A_h, \emptyset)$ from the set $\Gamma(G(E^*))$ which satisfies both conditions (a) and (b) and has the minimum objective function value

$$\Phi(C_1(G_h(E^*)), C_2(G_h(E^*)), \ldots, C_n(G_h(E^*)))$$

among all the digraphs $G_r(E^*) = (\mathcal{O}, A \cup A_r, \emptyset) \in \Gamma(G(E^*))$. The above digraph $G_h(E^*) = (\mathcal{O}, A \cup A_h, \emptyset) \in \Gamma(G(E^*))$ is said to be optimal for the problem $HJ|r_i|\Phi$ if the digraph $G_h(E^*)$ has the minimum objective function value Φ.

For illustration, we assume that there are two jobs $\mathcal{J} = \{J_1, J_2\}$, which must be processed on machines of two types, $\varphi = 2$. Thus, each of the two jobs has two operations: $\mathcal{O}_1 = \{Q^1_{1,1}, Q^2_{1,2}\}$, $\mathcal{O}_2 = \{Q^2_{2,1}, Q^1_{2,2}\}$. The chromosome $g^t = (q_1, q_2, q_3, q_4) = (1, 2, 3, 4)$ determines a unique semi-active schedule for this job-shop problem $J|r_i|\Phi$. Namely, operation $Q^1_{1,1}$ is processed before operation $Q^1_{2,2}$ on the machine of the type 1, and operation $Q^2_{1,2}$ is processed before operation $Q^2_{2,1}$ on the machine of the type 2.

If we generalize the above job-shop problem by assuming that there are two identical machines of the type 1 and two identical machines of the type 2, then we obtain the hybrid job-shop problem $HJ|r_i|\Phi$ with $\mathcal{M}_1 = \{M_1, M_2\}$ and $\mathcal{M}_2 = \{M_3, M_4\}$. Due to Lemma 5, the chromosome $g^t = (q_1, q_2, q_3, q_4) = (1, 2, 3, 4)$ determines a semi-active schedule for the problem $HJ|r_i|\Phi$, where both machines M_2 and M_4 are idle: $\mathcal{O}^1 = \mathcal{O}^1 \cup \emptyset$ and $\mathcal{O}^2 = \mathcal{O}^2 \cup \emptyset$. However, this chromosome g^t cannot determine the assignments of the operations to the different machines from the sets $\mathcal{M}_k$, $k \in \{1, 2\}$, for the problem $HJ|r_i|\Phi$.

5.2 The chromosome g^t_* for the problem $HJ|r_i|\Phi$

The chromosome g^t determines a heuristic solution for the job-shop problem $J|r_i|\Phi$ (one of such chromosomes is given in (4) for Example 1). To upgrade the chromosome g^t in order to determine any semi-active schedule S for the hybrid job-shop problem $HJ|r_i|\Phi$, the modified chromosome g^t_* must determine a partition of the operations $\mathcal{O}^k$ into $|\mathcal{M}_k|$ subsets. To determine this partition, we shall use $|\mathcal{M}_k| - 1$ delimiter instances of type d_k ($d_k = -k$), where k is equal to the type of the machines belonging to set $\mathcal{M}_k$.

Each delimiter $d_k = -k$ establishes the border between two subsets of operations from the set $\mathcal{O}^k$ assigned to different machines $M_u \in \mathcal{M}_k$ and $M_v \in \mathcal{M}_k$, $u \neq v$. To distinguish the delimiters from the genes q_i in the chromosome g^t_*, the delimiters d_k are negative integer numbers $d_k = -k$ while the genes q_i are natural numbers.

Let us consider Example 2 of the hybrid job-shop problem $HJ|r_i|\Phi$. Example 2 has the same input data as Example 1 of the job-shop problem $J|r_i|\Phi$ has, except the existence of parallel (identical) machines for each type 1, 2 and 3. If the parallel machines are as follows: $\mathcal{M}_1 = \{M_1, M_2\}$, $\mathcal{M}_2 = \{M_3\}$, $\mathcal{M}_3 = \{M_4, M_5, M_6\}$, then the chromosome g^t given in (4) can be modified into the following chromosome g^t_* for the hybrid job-shop problem $HJ|r_i|\Phi$:

$$\begin{aligned} g^t_* &= (q_1, q_{10}, q_7, -d_3, q_4, -d_1, q_2, q_{11}, q_5, q_8, q_{12}, -d_3, q_3, q_6, q_9) \\ &= (1, 10, 7, -3, 4, -1, 2, 11, 5, 8, 12, -3, 3, 6, 9). \end{aligned} \tag{5}$$

In the chromosome g^t_* given in (5), the numbers $d_1 = -1$ are delimiters for the subsets of the machine set $\mathcal{M}_1 = \{M_1, M_2\}$ of the type 1. The first delimiter $d_1 = -1$ expresses that all operations from the operation set $\mathcal{O}^1$, which are located in the chromosome g^t_* on the left-hand side from $d_1 = -1$, must be processed on machine M_1 and those located on the right-hand side from $d_1 = -1$ must be processed on machine M_2.

Since there is only one machine of the type 2, $\mathcal{M}_2 = \{M_3\}$, the delimiter $d_2 = -2$ is absent in the chromosome g^t_*.

As there are three machines of the type 3, $\mathcal{M}_3 = \{M_4, M_5, M_6\}$, two delimiters of the type $d_3 = -3$ are used in the chromosome g^t_* to determine three subsets of the set $\mathcal{O}^3$. All operations from the set

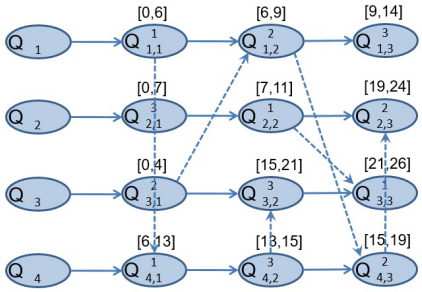

Figure 4: The digraph G'_h (without transitive arcs) achieved from a chromosome g^t_* for the hybrid job-shop problem $HJ|r_i|\Phi$.

$\mathcal{O}^3$, which are located in the chromosome g^t_* on the left-hand side from $d_3 = -3$, must be processed on machine M_4. All operations from the set $\mathcal{O}^3$, which are located in the chromosome g^t_* between the two delimiters of the type $d_3 = -3$, must be processed on machine M_5. All operations from the set $\mathcal{O}^3$, which are located in the chromosome g^t_* after both delimiters of the type $d_3 = -3$, must be processed on machine M_6. Figure 4 represents one of the digraphs $G_h(E^*) \in \Gamma(G(E^*))$ which are determined from the chromosome g^t_* given in (5).

5.3 The configuration of the genetic algorithm

The chromosome g^t_* as a parent, can introduce a heuristic solution for the hybrid job-shop problem $HJ|r_i|\Phi$. The simulator produces a digraph $G_h(E^*) = (\mathcal{O}, A \cup A_h, \emptyset) \in \Gamma(G(E^*))$ with $E^* \subset E$ based on the chromosome g^t_* that defines a semi-active schedule $S(G'_h(E^*))$. The objective function Φ will be used for evaluating the quality of the chromosome g^t_* to make a schedule $S(G_h(E^*))$. In the genetic algorithm, this objective function is known as the fitness function. The developed genetic algorithm tries to use its crossover and mutation operators to generate a better generation based on the current generation.

In each generation, the size of the population was set to 300. In each crossover operation, randomly 50% to 90% of a genome in an offspring is selected from the first parent and the rest is selected from the second parent. This random selection helps the genetic crossover operator to produce offspring with a different rate of similarity to the parents. The probability of mutation was 0.05%. In each generation, 70% of these individuals are results of the crossover operator. The mutation operator produces 10% of the chromosomes, and the remaining offspring are elites from the previous generation.

The crossover operator is a one-point crossover, which works as follows. In the first step, the genetic algorithm selects randomly two parents g^x_* and g^y_* from the current generation in order to generate their offspring g^z_*. The chromosome g^x_* acts as the main chromosome and a cutting point must be selected in this chromosome. Without any manipulation, the sequence of genes before this cutting point will be selected from the chromosome g^x_* and will make the first segment of the chromosome g^z_* that must be constructed.

In the next step, the genetic algorithm starts to scan the chromosome g^y_* and for each gene q_a, which is met in the chromosome g^y_*, the constructed part of the chromosome g^z_* is checked. If the gene q_a was not added to the chromosome g^z_* yet, the genetic algorithm appends the gene q_a to the

end of the constructed part of the chromosome g_*^z. Otherwise, the gene q_a is ignored in order to avoid a repetition of the gene q_a in the chromosome g_*^z.

If the algorithm meets a delimiter of type $d_k = -k$, then the number of the delimiters of the type $d_k = -k$ which are already inserted into the chromosome g_*^z is compared with the maximal number $|\mathcal{M}_k| - 1$ of the permissible delimiter of type $d_k = -k$. If the number of the delimiters of the type $d_k = -k$ is already equal to $|\mathcal{M}_k| - 1$ in the constructed part of the chromosome g_*^z, then the delimiter of type $d_k = -k$, which is met in the chromosome g_*^y, is not included into the chromosome g_*^z. Finally, an offspring g_*^z will be produced such that the first segment of the chromosome g_*^z belongs to the first parent g_*^x and the second segment of the chromosome g_*^z belongs to the second parent g_*^y. The constructed chromosome g_*^z inherits the orders of the genes from both parents, namely: in the first segment of the chromosome g_*^z, the sequence of genes is copied from the chromosome g_*^x and in the second part of the chromosome g_*^z, the sequence of genes is copied from the chromosome g_*^y.

As an example, we consider the chromosome $g_*^x = (q_1 = 1, q_{10} = 10, q_7 = 7, d_3 = -3, q_4 = 4, d_1 = -1, q_2 = 2, q_{11} = 11, q_5 = 5, q_8 = 8, q_{12} = 12, d_3 = -3, q_3 = 3, q_6 = 6, q_9 = 9)$ and the chromosome $g_*^z = (q_1 = 1, q_7 = 7, q_2 = 2, q_4 = 4, d_1 = -1, q_8 = 8, d_3 = -3, q_1 = 1, q_6 = 6, q_{11} = 11, q_5 = 5, q_{12} = 12, d_3 = -3, q_9 = 9, q_2 = 2)$, which have been selected randomly from the current generation as parents to produce a new offspring. Let the genetic algorithm find the cutting point $q_5 = 5$ in the main chromosome g_*^x. Therefore, the first segment of the offspring will be as follows: $(q_1 = 1, q_{10} = 10, q_7 = 7, d_3 = -3, q_4 = 4, d_1 = -1, q_2 = 2, q_{11} = 11, q_5 = 5)$. Then, the chromosome g_*^y will be scanned to find the order of the remaining genes in the chromosome g_*^y that must be copied in the second segment of the chromosome g_*^z. This order of the remaining genes is as follows: $(q_8 = 8, q_6 = 6, q_{12} = 12, d_3 = -3, q_9 = 9, q_3 = 3)$. Then these two segments will be concatenated to produce the offspring chromosome as follows: $g_*^z = (q_1 = 1, q_{10} = 11, q_7 = 7, d_3 = -3, q_4 = 4, d_1 = -1, q_2 = 2, q_{11} = 11, q_5 = 5, q_8 = 8, q_6 = 6, q_{12} = 12, d_3 = -3, q_9 = 9, q_3 = 3)$.

The mutation operator selects a chromosome g_*^t randomly. Then, two numbers a and b are selected randomly in the chromosome $g_*^t = (\ldots, a, \ldots, b, \ldots)$. After this mutation, due to a swapping of the selected numbers, we obtain the following chromosome $g_*^t = (\ldots, b, \ldots, a, \ldots)$, which is the same as the chromosome $g_*^t = (\ldots, a, \ldots, b, \ldots)$ except the positions of the numbers a and b. Each of the two numbers a and b may be either positive (i.e., a gene) or negative, (i.e., a delimiter). For example, we consider the chromosome $g_*^z = (q_1 = 1, q_{10} = 11, q_7 = 7, d_3 = -3, q_4 = 4, d_1 = -1, q_2 = 2, q_{11} = 11, q_5 = 5, q_8 = 8, q_6 = 6, q_{12} = 12, d_3 = -3, q_9 = 9, q_3 = 3)$. Let the selected numbers a and b be as follows: $a = d_3 = -3$ and $b = q_6 = 6$. After the mutation of the chromosome g_*^z, we obtain the following chromosome: $g_*^{z'} = (q_1 = 1, q_{10} = 11, q_7 = 7, q_6 = 6, q_4 = 4, d_1 = -1, q_2 = 2, q_{11} = 11, q_5 = 5, q_8 = 8, d_3 = -3, q_{12} = 12, d_3 = -3, q_9 = 9, q_3 = 3)$.

Each constructed chromosome g_*^z defines a digraph $G_z = (\mathcal{O}, A \cup A_z, \emptyset) \in \Gamma(G)$. To define a semi-active schedule for the problem $HJ|r_i|\Phi$, the digraph G_z must be circuit-free (see Theorem 1). The condition (b) of Theorem 1 is satisfied due to the genetic algorithm, where the set of operations $\mathcal{O}^k$ is partitioned into at most φ_k subsets. However, since the above genetic algorithm uses crossover and mutation operators, there is a possibility to construct a circuit in the digraph G_z, i.e., condition (a) of Theorem 1 may be violated. Such a circuit in the digraph G_z corresponds to deadlocks in using resources and must be avoided. Two strategies can be used to avoid a circuit in the digraph G_z. The first strategy is to reject the chromosome g_*^z generating a circuit in the digraph G_z. This strategy means that, if the digraph G_z has a circuit, the chromosome g_*^z must be rejected and a new offspring must be generated instead of chromosome g_*^z. This strategy is simple but not efficient since the genetic algorithm wastes time to create a chromosome g_*^z and then to

check whether the digraph G_z is circuit-free.

The second strategy is based on a modification of the chromosome in such a way that the circuit can be removed from the digraph G_z. To remove the circuit, the orientation of any arc $(Q_{ij}^{\mu(ij)}, Q_{uv}^{\mu(uv)})$ with $\mu(ij) = \mu(uv)$, in the circuit must be changed to the opposite one $(Q_{uv}^{\mu(uv)}, Q_{ij}^{\mu(ij)})$. However, the question is: which arc is better to modify? In the developed algorithm, the following heuristic was used. In the digraph $G_z = (\mathcal{O}, A \cup A_z, \emptyset) \in \Gamma(G)$, while replacing the edge $[Q_{ij}^{\mu(ij)}, Q_{uv}^{\mu(uv)}] \in E$ either by the arc $(Q_{ij}^{\mu(ij)}, Q_{uv}^{\mu(uv)})$ or by the arc $(Q_{uv}^{\mu(uv)}, Q_{ij}^{\mu(ij)})$, the simulator checks whether the ending vertex $Q_{uv}^{\mu(uv)}$ or $Q_{ij}^{\mu(ij)}$ is accessible by itself or not? If not, the algorithm continues its processing on the next two conflict operations. Otherwise, if the ending vertex is accessible by itself by adding the arc $(Q_{ij}^{\mu(ij)}, Q_{uv}^{\mu(uv)})$ or the arc $(Q_{uv}^{\mu(uv)}, Q_{ij}^{\mu(ij)})$, this means that a circuit will appear in the digraph G_z. To fix the digraph, the direction of the arc $(Q_{ij}^{\mu(ij)}, Q_{uv}^{\mu(uv)})$ or the arc $(Q_{uv}^{\mu(uv)}, Q_{ij}^{\mu(ij)})$ would be reversed. This inversion will remove the circuit in the digraph.

6 Computational results

To evaluate the efficiency of the developed genetic algorithm (Algorithm GA) for the problem $HJ||C_{max}$ and the genetic algorithm (Algorithm GA-$\sum C_i$) for the problem $HJ||\sum C_i$, the benchmark instances for the problem $J||C_{max}$ introduced in [26] (the instances la16, la17, ..., la20) and [4] (the instances orb1, orb2, ..., orb10) have been generalized to the problems $HJ||C_{max}$ and $HJ||\sum C_i$. To this end, it was assumed that several identical machines of the same type are available in the processing system.

The numbers $|\mathcal{M}_k|$ of the identical machines of the type k have been randomly selected from the set $\{2, 3, 4, 5\}$. In Table 1, the assigned values for the number of available parallel machines of the tested benchmark instances for the problems $HJ||C_{max}$ and $HJ||\sum C_i$ are presented. The obtained randomly generated instances of the problem $HJ||C_{max}$ are denoted as la16pm, la17pm, ..., la20pm and orb1pm, orb2pm, ..., orb10pm, where the addition *pm* is used to indicate the instances of the hybrid job-shop problem with the makespan criterion. The corresponding instances of the problem $HJ||\sum C_i$ are denoted as la16ps, la17ps, ..., la20ps and orb1ps, orb2ps, ..., orb10ps, where the addition *ps* is used to indicate the hybrid job-shop problem with the minimization of the sum of the jobs completion times.

Note that, to evaluate each chromosome, a mixed graph must be generated and tested. Then, the critical path must be calculated to find the objective function C_{max} for the problem $HJ||C_{max}$. All C_i-values have to be calculated to find the value $\sum C_i$ for the problem $HJ||\sum C_i$. This process is rather time consuming and therefore, the number of chromosomes and number of generations must be selected carefully to have an efficient program realizing the genetic algorithm. In our computational experiments, the evaluation started from 200 chromosomes and 200 generations. For tuning the genetic program, first we started to change the number of chromosomes in steps of 50. Then we tried to increase the number of generations. Finally, it was observed that a combination of 300 chromosomes and 500 generations was the best tuning.

The generated benchmark instances for the problems $HJ||C_{max}$ and $HJ||\sum C_i$ have been solved by the branch-and-bound Algorithms B&B and B&B-$\sum C_i$ developed in [35]. In addition, the results obtained by Algorithm GA were compared with those obtained by Algorithm SOICT, Algorithm

Table 1: The numbers $\varphi_k = |\mathcal{M}_k|$ of identical machines in the instances $HJ||\Phi$ with $\Phi \in \{C_{max}, \sum C_i\}$.

Problem	Size $n \times \varphi$ of the the problem $J\|\|\Phi$	$\|\mathcal{M}_1\| = \varphi_1$	$\|\mathcal{M}_2\| = \varphi_2$	$\|\mathcal{M}_3\| = \varphi_3$	$\|\mathcal{M}_4\| = \varphi_4$	$\|\mathcal{M}_5\| = \varphi_5$	$\|\mathcal{M}_6\| = \varphi_6$	$\|\mathcal{M}_7\| = \varphi_7$	$\|\mathcal{M}_8\| = \varphi_8$	$\|\mathcal{M}_9\| = \varphi_9$	$\|\mathcal{M}_{10}\| = \varphi_{10}$
1	2	3	4	5	6	7	8	9	10	11	12
Problem $HJ\|\|C_{max}$											
orb1pm	10×10	4	5	3	2	4	3	4	2	5	5
orb2pm	10×10	4	2	4	3	5	2	3	3	4	2
orb3pm	10×10	4	4	5	3	5	4	3	3	4	3
orb4pm	10×10	4	3	3	2	4	2	3	2	3	4
orb5pm	10×10	4	5	4	2	5	4	4	3	5	4
orb6pm	10×10	4	3	2	4	2	2	5	3	5	3
orb7pm	10×10	3	4	3	4	4	3	4	3	5	3
orb8pm	10×10	4	4	3	5	3	4	2	4	3	5
orb9pm	10×10	3	2	3	4	2	2	5	2	2	4
orb10pm	10×10	3	4	5	5	5	3	3	5	4	3
la16pm	10×10	2	2	5	2	2	3	4	3	2	3
la17pm	10×10	4	3	2	2	3	2	3	2	4	3
la18pm	10×10	4	2	2	3	4	2	2	3	2	3
la19pm	10×10	4	5	3	4	2	3	2	2	4	3
la20pm	10×10	2	3	3	2	5	2	3	3	4	2
Problem $HJ\|\|\sum C_i$											
orb1ps	10×10	3	4	5	4	5	3	5	5	4	3
orb2ps	10×10	3	2	3	2	5	4	5	3	4	5
orb3ps	10×10	4	3	5	3	5	4	5	3	4	4
orb4ps	10×10	5	5	5	5	5	5	5	5	5	5
orb5ps	10×10	4	5	4	3	5	3	4	5	5	4
orb6ps	10×10	3	4	3	4	4	3	5	3	5	3
orb7ps	10×10	3	5	4	3	5	3	5	4	3	3
orb8ps	10×10	5	5	5	5	5	5	5	5	5	5
orb9ps	10×10	5	3	5	4	4	3	5	3	4	3
orb10ps	10×10	4	4	5	5	5	4	5	5	5	4
la16ps	10×10	3	4	2	4	2	4	5	3	2	3
la17ps	10×10	4	3	3	2	3	3	4	2	4	3
la18ps	10×10	3	2	2	3	4	2	2	3	2	2
la19ps	10×10	2	5	2	4	2	3	2	2	3	3
la20ps	10×10	2	2	3	2	5	2	2	3	4	2

DCA and Algorithm PMJS-H developed in [36]. Next, we describe the schemes of these three heuristic algorithms.

Algorithm SOICT is based on sequencing the operations in a non-decreasing order of their completion times. The first stage of Algorithm SOICT consists in sequencing the jobs in the job-shop problem $J|r_i|C_{max}$ or the job-shop problem $J|r_i|\sum C_i$, respectively. The second stage is an assignment of the operations to the parallel machines, where Algorithm SOICT tries to parallelize the operations by considering the completion times $c(Q_{ij}^{\mu(ij)})$ of the operations $Q_{ij}^{\mu(ij)} \in \mathcal{O}^k$, which need to be processed on machines of the same type $k \in \{1, 2, \ldots, \varphi\}$. For binding the operations to the pool of parallel machines, the algorithm sequences the operations $Q_{ij}^{\mu(ij)}$ from the set $\mathcal{O}^k$ with $\mu(ij) = k$ in non-decreasing order of their completion times $c(Q_{ij}^{\mu(ij)})$. After sequencing the operations, a scheduler tries to bind the operations respecting the obtained sequence to the parallel machines, which are currently idle.

Algorithm DCA is based on the deletion of critical arcs (an arc belonging to the critical path in the digraph is called a critical arc). The first stage of Algorithm DCA is the same as the

first stage of Algorithm SOICT. In the second stage, Algorithm DCA considers critical arcs in the subgraphs $G_h^k(E^*) = (\mathcal{O}^k, A_h^k(E^*), \varnothing)$ of the weighted digraph $G_h(E^*) = (\mathcal{O},\ A \bigcup A_h, \varnothing) \in \Gamma(G(E^*))$. Algorithm DCA tries to delete the critical arcs if a specific condition is satisfied. After deleting a critical arc from the weighted digraph $G_h(E^*) = (\mathcal{O},\ A \bigcup A_h, \varnothing)$, the objective function value $\Phi(S(G_h(E^*)))$ may be decreased. Algorithm DCA deletes the critical arcs between the operations and distributes the operations from the set $\mathcal{O}^k$ to different machines from the set $\mathcal{M}_k$. After each deletion of a conflict arc, Algorithm DCA recalculates the critical paths in the obtained digraph and tries to find the next critical arc belonging to a new critical path in order to delete this arc.

Algorithm PMJS-H realizes the assignment of the operations $\mathcal{O}^k$ to the parallel machines of the type k and the sequencing the operations on the assigned machine simultaneously. Algorithm PMJS-H uses the conflict resolution strategy, i.e., the edges of the mixed graph $G = (\mathcal{O}, A, E)$ treated by Algorithm PMJS-H are a conflict in the following sense. An edge $[Q_{ij}^{\mu(ij)} Q_{uv}^{\mu(uv)}] \in E$ in the mixed graph $G = (\mathcal{O}, A, E)$ is a conflict edge if both orientations $(Q_{ij}^{\mu(ij)}, Q_{uv}^{\mu(uv)})$ and $(Q_{uv}^{\mu(uv)},\ Q_{ij}^{\mu(ij)})$ of this edge in the weighted digraph obtained from the mixed graph G lead to an increase either of the starting time $s(Q_{ij}^{\mu(ij)})$ of the operation $Q_{ij}^{\mu(ij)}$ or of the starting time $s(Q_{uv}^{\mu(uv)})$ of the operation $Q_{uv}^{\mu(uv)}$.

For generating a sequence of the operations $\mathcal{O}^*$ for the hybrid job-shop problem, a scheduler generates a sequence of the operations from the set $\mathcal{O}^*$ one by one on the machines from the set $\mathcal{M}$ using a breadth first search (i.e., the sibling vertexes in the digraph $(\mathcal{O},\ A, \varnothing)$ are considered before considering the child vertexes in the solution tree). For treating the operation $Q_{ij}^{\mu(ij)}$, it is checked if there is an idle machine in the set $\mathcal{M}_k$ to process this operation at the current time t or not? If yes, the operation $Q_{ij}^{\mu(ij)}$ is assigned to that machine. If there is no idle machine in the set $\mathcal{M}_k$ at time t, then Algorithm PMJS-H chooses a machine M_g from the set $\mathcal{M}_k$ with the minimal completion time of the last operation $Q_{uv}^{\mu(uv)}$ already assigned to that machine.

The above Algorithms SOICT, DCA and PMJS-H adopted for solving the problem $HJ||\sum C_i$ with the criterion $\sum C_i$ are denoted as Algorithms SOICT-$\sum C_i$, DCA-$\sum C_i$ and PMJS-H-$\sum C_i$, respectively.

The optimal objective function values C_{max} and $\sum C_i$ for the benchmark instances calculated by Algorithms B&B and B&B-$\sum C_i$, respectively, are presented in column 4 of Table 2. The objective function values C_{max} and $\sum C_i$ calculated by Algorithm GA and Algorithm GA-$\sum C_i$ are given in column 5. The objective function values C_{max} (values $\sum C_i$) calculated by Algorithms PMJS-H, SOICT and DCA (Algorithms SOICT-$\sum C_i$, DCA-$\sum C_i$ and PMJS-H-$\sum C_i$) are given in columns 6, 7 and 8, respectively.

The optimal objective function values are given in bold face in Table 2. The best values of the objective functions obtained by Algorithms PMJS-H, SOICT, DCA, and GA (Algorithms SOICT-$\sum C_i$, DCA-$\sum C_i$, PMJS-H-$\sum C_i$, and GA-$\sum C_i$) are underlined in Table 2. The worst values of the objective functions obtained by Algorithms PMJS-H, SOICT, DCA, and GA (Algorithms SOICT-$\sum C_i$, DCA-$\sum C_i$, PMJS-H-$\sum C_i$, and GA-$\sum C_i$) are given in italics in Table 2. Since the worst values of the objective function $\sum C_i$ are obtained only by Algorithm DCA-$\sum C_i$, the worst values of the objective functions obtained by the other Algorithms PMJS-H-$\sum C_i$, SOICT-$\sum C_i$ and GA-$\sum C_i$ (except Algorithm DCA-$\sum C_i$) are given in parentheses.

As it is shown in Table 2, Algorithm GA (Algorithm GA-$\sum C_i$) could solve 7 instances of the

Table 2: The objective functions values C_{max} and $\sum C_i$ obtained by the exact and heuristic algorithms, which were tested.

Problem $HJ\|\|C_{max}$	Problem size $n \times \Sigma_{i=1}^{\varphi}\varphi_i$	Number φ of machine types	Algorithm B&B	Algorithm GA	Algorithm PMJS-H	Algorithm SOICT	Algorithm DCA
1	2	3	4	5	6	7	8
orb1pm	10×37	10	**695**	*779*	750	771	761
orb2pm	10×32	10	**625**	*744*	649	727	734
orb3pm	10×38	10	**648**	**648**	669	736	*802*
orb4pm	10×30	10	**753**	**753**	809	815	*821*
orb5pm	10×40	10	**584**	619	609	*706*	612
orb6pm	10×33	10	**715**	723	768	*802*	763
orb7pm	10×36	10	**275**	*365*	290	327	312
orb8pm	10×37	10	**573**	**573**	601	*660*	631
orb9pm	10×29	10	**659**	681	701	*794*	761
orb10pm	10×40	10	**681**	**681**	*801*	772	722
la16pm	10×28	10	**717**	728	**717**	*810*	764
la17pm	10×24	10	**646**	**646**	669	*689*	**646**
la18pm	10×27	10	**663**	684	**663**	*749*	671
la19pm	10×32	10	**617**	**617**	624	*664*	652
la20pm	10×29	10	**756**	**756**	757	*830*	776

Problem $HJ\|\|\sum C_i$			Algorithm B&B-$\sum C_i$	Algorithm GA-$\sum C_i$	Algorithm PMJS-H-$\sum C_i$	Algorithm SOICT-$\sum C_i$	Algorithm DCA-$\sum C_i$
orb1ps	10×41	10	**5485**	5930	5859	(5956)	*6840*
orb2ps	10×36	10	**5442**	(5679)	5457	5618	*6613*
orb3ps	10×40	10	**5370**	5542	5413	(5893)	*6398*
orb4ps	10×50	10	**5725**	**5725**	5754	(5877)	*6552*
orb5ps	10×42	10	**5019**	5089	5107	(5482)	*5983*
orb6ps	10×37	10	**5709**	5926	5898	(5986)	*7201*
orb7ps	10×38	10	**2252**	(2492)	2466	2483	*3154*
orb8ps	10×50	10	**4626**	4661	4703	(4820)	*5452*
orb9ps	10×39	10	**5292**	5326	5371	(5414)	*5981*
orb10ps	10×46	10	**5673**	5702	5685	(5750)	*6920*
la16ps	10×32	10	**5553**	(5992)	5703	5984	*6434*
la17ps	10×31	10	**4847**	5078	4921	(5139)	*5761*
la18ps	10×25	10	**5315**	5608	5516	(5905)	*6638*
la19ps	10×28	10	**5461**	(5651)	5481	5628	*6059*
la20ps	10×27	10	**5551**	(6092)	5650	6068	*6814*

Table 3: Numbers of optimal, best and worst schedules constructed for problem $HJ||C_{max}$ by the tested heuristic algorithms.

Problem $HJ\|\|C_{max}$	Algorithm GA	Algorithm PMJS-H	Algorithm SOICT	Algorithm DCA
1	2	3	4	5
Numbers of optimal schedules	7	2	0	1
Numbers of best schedules	9	6	0	1
Numbers of worst schedules	3	1	9	2
Average error	3.99%	4.88%	13.33%	8.82%

problem $HJ||\Phi$ to optimality for the $\Phi = C_{max}$ criterion (and one instance for the $\Phi = \sum C_i$ criterion). The number of instances with the C_{max} criterion optimally solved by Algorithms PMJS-H, SOICT and DCA are 2, 0 and 1, respectively.

There is no instance with the $\sum C_i$ criterion, which was optimally solved by the tested heuristic algorithm. Algorithm GA (Algorithm GA-$\sum C_i$) could get 9 times the best objective function values among the heuristic algorithms tested for the C_{max} criterion (and 4 times for the $\sum C_i$ criterion). For the benchmark instances with the C_{max} criterion, the average percentage deviation of the objective function value from the optimal one is equal to 3.99% for Algorithm GA (see last row of Table 3). For Algorithms PMJS-H, SOICT and DCA, these numbers are equal to 4.88%, 13.33% and 8.82%, respectively.

These numbers are presented in Table 3 along with the numbers of the optimal, best and worst schedules constructed by the tested heuristic algorithms. This experimental comparison on the tested benchmark instances shows the efficiency of the developed genetic Algorithm GA for solving the problem $HJ||C_{max}$. Algorithm GA outperforms Algorithms SOICT and DCA for all four factors presented in Table 3 while solving the problem $HJ||C_{max}$. Algorithm GA outperforms also Algorithm PMJS-H with the exception that Algorithm GA constructed 3 times the worst schedule (among the four tested heuristic algorithms) while Algorithm PMJS-H constructed one worst schedule. It should be noted that Algorithm PMJS-H needs less CPU-time than Algorithm GA for solving the same problem $HJ||C_{max}$.

For the $\sum C_i$ criterion, the average percentage deviation of the objective function value from the optimal one is equal to 4.32% for Algorithm GA-$\sum C_i$. For Algorithms PMJS-H-$\sum C_i$, SOICT-$\sum C_i$ and DCA-$\sum C_i$, the average percentage deviations are equal to 2.43%, 6.24% and 20.75 %, respectively. These numbers are presented in Table 4 along with the numbers of the optimal and best schedules constructed by the heuristic algorithms. The worst values of the objective function $\sum C_i$ obtained by Algorithms PMJS-H-$\sum C_i$, SOICT-$\sum C_i$ and GA-$\sum C_i$ (without Algorithm DCA-$\sum C_i$) are given in the second-to-last row of Table 4. Therefore, Algorithm GA-$\sum C_i$ outperforms each of the two Algorithms SOICT-$\sum C_i$ and DCA-$\sum C_i$ for all four factors presented in Table 4. However, Algorithm PMJS-H-$\sum C_i$ outperforms each of the three Algorithms SOICT-$\sum C_i$, DCA-$\sum C_i$ and GA-$\sum C_i$ for all four factors presented in Table 3 with the exception, where Algorithm GA-$\sum C_i$ obtained an optimal schedule while Algorithm PMJS-H-$\sum C_i$ did not obtain an optimal

Table 4: Numbers of optimal, best and worst schedules constructed for problem $HJ||\sum C_i$ by the tested heuristic algorithms.

Problem $HJ\|\|\sum C_i$	Algorithm GA-$\sum C_i$	Algorithm PMJS-H-$\sum C_i$	Algorithm SOICT-$\sum C_i$	Algorithm DCA-$\sum C_i$
1	2	3	4	5
Numbers of optimal schedules	1	0	0	0
Numbers of best schedules	4	11	0	0
Numbers of worst schedules	0	0	0	15
Numbers of worst schedules obtained by Algorithms GA-$\sum C_i$, PMJS-H-$\sum C_i$ and SOICT-$\sum C_i$	5	0	10	-
Average error	4.32%	2.43%	6.24%	20.75%

schedule at all.

Note that Algorithm PMJS-H-$\sum C_i$ needs less CPU-time than Algorithm GA-$\sum C_i$ for solving the same problem $HJ||\sum C_i$.

7 Conclusion

In Sections 3 – 5, genetic algorithms, GA and GA-$\sum C_i$, were provided for solving the hybrid job-shop problem $HJ|r_i|\Phi$. Two regular criteria have been considered: the minimization of the makespan, $\Phi = C_{\max}$, and the minimization of the sum of the job completion times, $\Phi = \sum C_i$. In the problem $HJ|r_i|\Phi$, several identical machines $\mathcal{M}_k \neq \emptyset$ of the type $k \in \{1, 2, \ldots, \varphi\}$ are available for processing the corresponding subset $\mathcal{O}^k$ of the operation set. This flexibility allows a scheduler to reduce the objective function value since any operation $Q_{ij}^{\mu(ij)} \in \mathcal{O}^k$ may be processed on any machine of the type $k = \mu(ij)$ at the stage j of the job J_i. So, the operation set $\mathcal{O}^k$ may be partitioned into $|\mathcal{M}_k|$ subsets for their simultaneous processing. As a matter of fact, the problem of finding an optimal schedule for processing the given operation set $\mathcal{O}$ in the job-shop is combined with an assignment of the operations $\mathcal{O}^k$ to the machines in the set $\mathcal{M}_k$ for each type $k \in \{1, 2, \ldots, \varphi\}$.

The developed genetic algorithms for solving the problem $HJ|r_i|\Phi$ use a special chromosome based on the mixed graph model for solving the above subproblems simultaneously. These chromosomes had the ability to partition the operations into the set $\mathcal{O}^k$ and to represent a whole solution.

The computational results for the benchmark instances (Section 6) showed that the developed genetic algorithm is more efficient for problem $HJ|r_i|C_{\max}$ than the other three heuristic algorithms being tested. For the problem $HJ|r_i|\sum C_i$, the developed genetic algorithm is more efficient than

other two heuristic algorithms being tested but less efficient than Algorithm PMJS-H-$\sum C_i$.

For future research, it is recommended to develop similar genetic algorithms for the problem $HJ|r_i|\Phi$ with harder criteria like the minimization of job tardiness. Another research direction is to work on blocking and limited buffers in the hybrid job-shop problem. The consideration of no-wait jobs is another interesting research subject for the hybrid job-shop with a possible application to the train timetabling problem for a multi-track railroad system.

References

[1] M. Abdolzadeh, and H. Rashidi, An approach of cellular learning automata to job-shop scheduling problem, International Journal of Simulation: Systems, Science and Technology, 11, 2010, 56-64.

[2] J. Adams, E. Balas, and D. Zawack, The shifting bottleneck procedure for jobshop scheduling. Management Science, 34(3), 1988, 391-401.

[3] N. Al-Hinai, T.Y. ElMekkawy, Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm, International Journal of Production Economics, 132(2), 2011, 279-291.

[4] D. Applegate, and W. Cook, A computational study of the job-shop scheduling problem. ORSA Journal on Computing, 3, 1991, 149-156.

[5] A. Bagheri, M. Zandieh, I. Mahdavi, M. Yazdani, An artificial immune algorithm for the flexible job-shop scheduling problem. Future Generation Computer Systems, 26, 2010, 533-541.

[6] E. Balas, Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. Operations Research, 17(6), 1969, 941-957.

[7] P. Brucker, B. Jurisch, and B. Sievers, A branch and bound algorithm for the job-shop scheduling problem Discrete Applied Mathematics, 49, 1994, 107-127.

[8] P. Brucker, Y.N. Sotskov, and F. Werner, Complexity of shop-scheduling problems with fixed number of jobs: a survey, Mathematical Methods of Operations Research, 65, 2007, 461-481.

[9] J.C. Chen, C. Wu, C. Chen, K. Chen, Flexible job shop scheduling with parallel machines using Genetic Algorithm and Grouping Genetic Algorithm. Expert Systems with Applications, 39(11), 2012, 10016-10021.

[10] T.C. Chiang, H.J. Lin, 2012. A simple and effective evolutionary algorithm for multi objective flexible job-shop scheduling. International Journal of Production Economics, 141(1), 2012, 87-98.

[11] U. Dorndorf, E. Pesch, Evaluation based learning in a job-shop scheduling environment, Computers & Operations Research, 22, 1995, 25-40.

[12] G. Ferrer, N. Dew, U. Apte, Analyzing the effect of dispatching rules on the scheduling performance through grammar based flexible scheduling system. International journal of production economics, 124(2), 2010, 369-382.

[13] J. Gao, L. Sun, M. Gen, M., A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Computers & Operations Research, 35, 2008, 2892-2907.

[14] T. Gabel, M. Riedmiller, Adaptive reactive job-shop scheduling with reinforcement learning agents International Journal of Information Technology and Intelligent Computing, 2, 2007, 1-30.

[15] M.R. Garey, D.S. Johnson, R. Sethi, The complexity of the flowshop and jobshop scheduling. Mathematics of Operational Research, 1, 1976, 117-129.

[16] C.D. Geiger, R. Uzsoy, and H. Aytug, 2006. Rapid modelling and discovery of priority dispatching rules: an autonomous learning approach, Journal of Scheduling, 9, 2006, 7-34.

[17] M. Gen, L. Lin, H. Zhang, Evolutionary techniques for optimization problems in integrated manufacturing system: State-of-the-art-survey. Computers & Industrial Engineering, 56, 2009, 779-808.

[18] O. Gholami, and Y.N. Sotskov, Solving parallel machines job-shop scheduling problems by an adaptive algorithm, International Journal of Production Research 52(13), 2014, 3888-3904.

[19] O. Gholami, and Y.N. Sotskov, A neural network algorithm for servicing jobs with sequential and parallel machines, Automation and Remote Control 75(7), 2014, 1203-1220.

[20] F. Glover Tabu search – part 1, 1989. ORSA Journal on Computing, 1(2), 1989, 190–206.

[21] R.L. Graham, E.R. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics, 5, 1979, 287-326.

[22] R. Haupt, A survey of priority rule-base scheduling, OR Spectrum, 11, 1989, 3-16.

[23] A.B. Hmidaa, M. Haouarid, M.-J. Hugueta, P. Lopez, Discrepancy search for the flexible job shop scheduling problem, Computers and Operations Research, 37(12), 2010, 2192-2201.

[24] J. Lange and F. Werner, Approaches to modeling train scheduling problems as job-shops with blocking constraints, Preprint 18/15, Faculty of Mathematics, Otto-von-Guericke-University, 2015, 25 pages.

[25] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, Sequencing and scheduling: Algorithms and complexity, Handbooks in Operations Research and Management Science 4, 1993, 445-522.

[26] S. Lawrence, Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Graduate School of Industrial Administration. Carniege-Mellon University, Pittsburgh, P.A., 1994.

[27] D.C. Li, I.S. Shi, 1994. Using unsupervised learning technologies to induce scheduling knowledge for FMSs, International Journal of Production Research, 32, 1994, 2187-2199.

[28] J. Li, Q. Pan, S. Xie, An effective shuffled flog-leaping algorithm for multi-objective flexible job shop scheduling problems. Applied Mathematics and Computation, 218, 9353-9371.

[29] S.Q. Liu, E. Kozan, Scheduling trains as a blocking hybrid job shop scheduling problem. Computers and Operations Research, 36(10), 2009, 2840-2852.

[30] R.M. Lusby , J. Larsen, M. Ehrgott, D. Ryan, Railway track allocation: Models and methods. OR Spektrum, 33, 2011, 843-883.

[31] J.O. Ong and I. Maulana, Model development of shifting bottleneck heuristic to solve job-shop scheduling problem with parallel machines. International Conference on Engineering of Tarumanagara (ICET 2013). October, 2(3), 2013.

[32] S.S. Panwalkar, W. Iskander, 1977. A survey of scheduling rules. Operations Research, 25(1), 1977, 45-61.

[33] J. Paulli, A hierarchical approach for the FMS scheduling problem. European Journal of Operational Research, 86(1), 1995, 32-42.

[34] A. Rossi, E. Boschi, A hybrid heuristic to solve the parallel machines job-shop scheduling problem, Advances in Engineering Software, 40(2), 2009, 118-127.

[35] Y.N. Sotskov, Mixed multigraph approach to scheduling jobs on machines of different types. Optimization, 42(3), 1997, 245-280.

[36] Y.N. Sotskov, and O. Gholami, Mixed graph model and algorithms for hybrid job-shop scheduling problems. International Journal of Production Research. In press, 2015.

[37] V.S. Tanaev, Y.N. Sotskov, V.A. Strusevich, Scheduling Theory: Multi-Stage Systems. Kluwer Academic Publishers, Dordrecht, Netherlands, 1994.

[38] S. Wang, J. Yu, An effective heuristic for flexible job-shop scheduling problem with maintenance activities. Computers and Industrial Engineering, 59(3), 2010, 436-447.

[39] F. Werner, A Survey of Genetic Algorithms for Shop Scheduling Problems in: P. Siarry (ed.), Heuristics: Theory and Applications, Nova Science Publishers, 2013, 161-222.

[40] L.N. Xing, Y.W. Chen, and K.W. Yang, Multi-objective flexible job shop schedule: Design and evaluation by simulation modelling, Applied Soft Computing, 9(1), 2009, 362-376.