A Neighborhood Search Function for Flexible Job Shop Scheduling with Separable Sequence-Dependent Setup Times

Tamer F. Abdelmaguid

Department of Mechanical Design and Production,
Faculty of Engineering, Cairo University,
Giza 12613, Egypt
Tel.: +20106-2689333
Fax: +202-35693025
E-mail: tabdelmaguid@eng.cu.edu.eg

Abstract

This paper addresses the makespan minimization problem in scheduling flexible job shops whenever there exist separable sequence-dependent setup times. An extension to the neighborhood search functions of Mastrolilli and Gambradella, developed for the flexible job shop scheduling problem (FJSP), is provided. It is shown that under certain conditions such an extension is viable. Accordingly, a randomized neighborhood search function is introduced, and its best search parameters are determined experimentally using modified FJSP benchmark instances. A tabu search approach utilizing the proposed neighborhood search function is then developed, and experimentations are conducted using the modified instances to benchmark it against a lower bound. Experimental results show that on average, the tabu search approach is capable of achieving optimality gaps of below 10% for instances with low average setup time to processing time ratios.

Keywords: flexible job shop scheduling, sequence-dependent setup time, neighborhood search, tabu search, algorithm design

1. Introduction

The general job shop scheduling problem (JSP) was introduced to deal with the sequencing decisions of manufacutring operations in low volume-high variety manufacturing systems in which part routes are fixed while differ considerably from one part to the other. With the advent of flexible manufacturing systems (FMSs), it became necessary to relax the assumption of fixed part routes in order to be able to deal with the arising scheduling problems in these systems. FMSs are characterized by the existence of general purpose computer numerically controlled (CNC) machines that can perform various types of manufacturing operations. This made it possible for a manufacturing operation of a given part to be conducted on any of a set of different machines with possibly different processing times. This means that the operations' routing became a decision variable instead of being fixed. In the 1980s, the flexible job shop scheduling problem (FJSP) was introduced to address this situation.

The FJSP, denoted $FJc|rcrc|$ according to the three-field notation in [1], is composed of two interacting problems. The first is concerned with selecting a machine for each operation, and

the second is a regular JSP with possible recirculation, i.e. a given part may visit the same machine more than once. As indicated in [2], there are two approaches that are followed in dealing with this problem. The first is a hierarchical or a decomposition approach which solves the machine assignment problem first; and based on its solution, a JSP is formulated and solved. The second is a concurrent one which looks for a solution to both problems simultaneously. Early studies of the FJSP developed both hierarchical and concurrent approaches that are based on mixed integer linear programming (MILP) models and dispatching rules [3, 4, 5, 6]. In the early 1990s, many researchers found the FJSP fertile for emerging metaheuristic approaches since exact algorithms are computationally prohibitive and the capabilities of traditional heuristic approaches are limited. The metaheuristics that are frequently applied to the FJSP are tabu search (TS) and genetic algorithm (GA).

In the TS literature, a hierarchical TS approach for $FJc|rcrc|C_{\max}$ and $FJc|rcrc|\Sigma w_j T_j$ was developed in [2]. Simple neighborhood search rules adopted from the JSP literature are used for the scheduling part. Later, more sophisticated neighborhood search strategies were developed in [7] and used within a TS approach to solve the problem $FJc|rcrc|C_{\max}$ in which an operation's processing time is not machine dependent. Dauzère-pérès and Paulli [8] developed another neighborhood search structure that allows either changing the position of an operation in the processing sequence of its already assigned machine or placing it on the sequence of another machine. They used TS for solving the problem $FJc|rcrc|C_{\max}$ with their neighborhood search structure. Their comparison of results with [7] shows significant improvement. Later, Mastrolilli and Gambradella [9] developed two new neighborhood search functions for the same problem and were capable of achieving better results using TS. Up till now, the TS approach of Mastrolilli and Gambradella [9] remains a competitive method for the problem $FJc|rcrc|C_{\max}$.

In the GA literature, Mesghouni et al [10] are the first to use genetic (evolutionary) algorithm for solving the problem $FJc|rcrc|C_{\max}$. Different GA representations were then developed in [11, 12, 13, 14]. Later, Gao et al [15] developed a hybrid GA which integrates GA with local neighborhood search structures similar to that of Mastrolilli and Gambradella [9]. Their computational results indicate that their GA implementation is comparable to the best known TS implementation by Mastrolilli and Gambradella [9]. Pezzella et al [16] used the same GA coding in [13] with two additional methods for solving the machine assignment subproblem. However, the GA of Gao et al [15] demonstrates better results on standard benchmark instances. Wang et al [17] presented a similar GA coding with immune and entropy principle for solving a multi-objective FJSP. Recently, Zhang et al [18] proposed an improved GA representation which reduces the computational time and produces competitive results with new upper bounds for some standard benchmark instances. They demonstrated the importance of utilizing an efficient initialization method for a better performance of the GA.

Recently, new metaheuristic techniques were applied to the $FJc|rcrc|C_{\max}$ problem. The parallel variable neighborhood search technique in [19] utilizes the solution representation in [13] and implements six different neighborhood search operations within a parallel variable neighborhood search structure. Better results are achieved compared to the GA technique of Pezzella et al [16]; however their results are inferior to those of Gao et al [15]. Hmida et al [20] utilized a discrepancy search technique with extending the neighborhood search functions in [7]. Their results on standard benchmark instances show that their technique provides competitive results. Bożejko et al [21] presented hybrid metaheuristics working on two levels. The first level works on the machine selection decisions; while the second level solves the operations scheduling part of the problem. The second level uses tabu search in a parallel fashion that allows for the utilization of several processors simultaneously. They were capable of achieving new best solutions

on some benchmark instances. Recently, Yuan and Xu [22] developed a hybrid differential evolution algorithm for the $FJc|rcrc|C_{\max}$ problem, and reported competitive results on standard benchmark instances. They utilized the neighborhood search of Mastrolilli and Gambradella [9] for improving solutions at each iteration.

One of the important factors being addressed in the scheduling literature is the setup time needed to get a machine ready for the processing of a given part. Setup times are classified into separable and inseparable. Separable setups are conducted on the machine without the requirement of having the unprocessed part engaged. On the other hand, the unprocessed part must be available during inseparable setups. The time needed by the inseparable setup can be easily added to the processing time and the resultant summation can then be treated as the processing time in a traditional way. In many situations, separable setup times depend on the sequence by which parts are processed on a machine. Such sequence dependent-setup times are apparent in FMSs as they are related to tool changes between different parts. Recent literature reviews on the different types of scheduling and lot sizing problems with sequence-dependent setup times are provided in [23, 24].

This paper is concerned with the FJSP with separable sequence-dependent setup times (FJSP-SDST) with a single objective of minimizing the makespan, denoted $FJc|rcrc, s_{ijk}|C_{\max}$. As a generalization of the FJSP, the studied problem is known to be NP-hard which necessitates the development of efficient solution techniques. Few papers have addressed the FJSP-SDST. To the best of our knowledge, Choi and Choi [25] are the first to address this problem with the objective of minimizing the makespan. They provided an MILP model and developed a local search algorithm that can incorporate different dispatching rules. However, their experimental results showed that a tabu search algorithm adapted from the JSP literature demonstrates better performance in most of the test instances. Guimarães and Fernandes [26] developed a GA approach for the problem $FJc|rcrc, s_{ijk}|$ considering four different objectives simultaneously including the minimization of the makespan. Minor experiments were conducted using four instances from the FJSP literature in [13] with a modification that adds setup times.

Saidi-Mehrabad and Fattahi [27] developed a TS approach for the problem $FJc|rcrc, s_{ijk}|C_{\max}$ with a special structure that exactly specifies two machines for each operation. They compared the TS results for specially designed small size instances (up to 3 jobs and 3 machines) with the optimal solutions obtained by a commercial MILP solver. Their TS was capable of achieving optimal solutions for the small size instances; while, there is no clue about its relative performance with larger size instances. Defersha and Chen [28] developed a parallel GA approach for solving a variant of the FJSP-SDST in which jobs take the form of batches and a job can be moved to a next machine whenever a pre-specified number of parts are completed on the current machine. They used the GA coding of Kacem et al [13], and they conducted minor experiments to show the capability of using parallel search techniques in escaping local minima, especially with large-sized problem instances, when compared to a single-population GA.

Unlike the FJSP, the studied problem is relatively newly addressed in the literature and there is no standard set of benchmark instances that can be used to compare the different solution approaches. Moreover, there is an apparent inconsistency between the different contributions cited in the previous two paragraphs in terms of the problem structure and the experimentation policy for the developed solution approaches. Despite the fact that the studied problem is a clear extension to the FJSP, the works in [25, 27] failed to consider adapting some of the well-established FJSP solution approaches and applying them to the FJSP-SDST, and to develop modified set of benchmark instances based on FJSP instances. Other works in [26, 28] recognized that point; however they did not provide extensive experimentations that clearly identify the level of quality

for their developed approaches.

Since the success of the metaheuristic approaches for the FJSP, specifically the TS in [9] and the hybrid GA in [15], is dependent on the utilization of an efficient neighborhood search function, it is imperative to consider an adaptation to the FJSP neighborhood search function used therein which permits the extension of such metaheuristics to the FJSP-SDST. Therefore, the contribution of the current work lies on the extension of the successful neighborhood search function developed in [9] to the FJSP-SDST by studying theoretically the conditions based upon which such an extension is possible, and identifying some search parameters of that function at which the best performance is expected. Furthermore, modifications to selected FJSP test instances are introduced in an experimental design with the objective of studying the effects of the different factors related to the problem structure on the performance of the function. Then, a tabu search metaheuristic is developed utilizing the proposed neighborhood search function and its results are compared to a lower bound.

The rest of this paper is organized as follows. In Section 2, some mathematical notations on permutations and their operations are defined. Permutations are used to represent both processing sequences on machines and paths on the network representation of the studied problem which is described in Section 3. The main theory behind the neighborhood search function is presented in Section 4, followed by a description of the proposed neighborhood search function in Section 5. Experimentation required to determine the best working prameters of the neighborhood search function are provided in Section 6; followed by the description and results obtained for the developed tabu search approach in Section 7. Finally, the conclusion and directions for future research are provided in Section 8.

2. Notations for permutations and their operations

Permutations are used in this paper to represent processing sequences on machines as well as paths on the network representation of the studied problem. Since there is no standard mathematical notation for representing permutations and their operations, the following notations are adopted herein. We define a permutation as an ordered list of distinct elements represented as $\pi = (e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_n)$. For a given permutation π, $|\pi|$ denotes its size (number of elements), and $\pi[x]$ denotes the element located at position x. Element $\pi[x]$ is the same as element $\pi[y]$ only when $x = y$. A permutations may be defined in the form $\pi = (e_x$: logical conditions and/or relationships that specify each element e_x in position $x)$. An empty permutation is denoted (.), and we denote by $e \in \pi$ if element e is included in π and $e \notin \pi$ if not. The position of an element $e \in \pi$ is denoted $\xi(\pi, e)$ where $\xi(\pi, e) \in \{1, 2, \ldots |\pi|\}$. The set that contains all the elements of a permutation π is denoted $\mathbb{S}(\pi) = \{e$: $e \in \pi\}$.

For a given permutation π and a subset of consecutive indexes $P = \{r, r+1, r+2 \ldots\} \subseteq \{1, 2, \ldots |\pi|\}$, the permutation $\pi' = (e_x$: $e_x = \pi[y]$ and $x = y - r + 1$, $y \in P)$ is said to be sub-permutation of π and denoted $\pi' \subseteq \pi$. If $|\pi'| < |\pi|$, the sub-permutation relationship is represented by $\pi' \subset \pi$. In addition, the notation $\pi(r, q)$, where $1 \leq r \leq q \leq |\pi|$, is used to represent the sub-permutation $(e_x$: $e_x = \pi[y]$ and $x = y - r + 1$, $y \in \{r, r+1, \ldots q\})$; and $\pi(q, r) = (.)$. The intersection operation between two permutations, represented by $\pi_1 \cap \pi_2$, is the set of permutations $\{\pi' : \pi' \subseteq \pi_1$ and $\pi' \subseteq \pi_2\}$.

The appending operation is represented as $\pi \leftarrow \pi'$ which results in a new permutation that has the same sequence of elements as in π with the elements of π' added at the end with maintaining the sequence in π'. This operation is successful only under the condition that $\mathbb{S}(\pi) \cap \mathbb{S}(\pi') = \emptyset$;

otherwise, it returns (.). The insertion operation, represented as $\pi \overset{x}{\longleftarrow} \pi'$ where the insertion position $x \in \{0, 1, 2, ... |\pi|\}$, results in the permutation $(\pi_L \leftarrow \pi') \leftarrow \pi_R$, where π_L =($e_y : e_y$=$\pi[y]$, $y \in \{1, 2, ..., x\}$) if $x \geq 1$ and π_L =(.) otherwise; and π_R =($e_y : e_y$=$\pi[x + y]$, $y \in \{1, 2, ..., |\pi| - x\}$) if $x < |\pi|$ and π_R =(.) otherwise. Similar to the appending operation, the insertion operation is successful only under the condition that $\mathbb{S}(\pi) \cap \mathbb{S}(\pi') = \emptyset$; otherwise, it returns (.). In addition, we define the subtracting operation as removing permutation π' from permutation π, where $\pi' \subset \pi$. This operation is represented as $\pi - \pi' = \pi_L \leftarrow \pi_R$. Here, both π_L and $\pi_R \subset \pi$, and they are defined such that π=$(\pi_L \leftarrow \pi') \leftarrow \pi_R$.

In addition, a special data structure is used for managing local selection alternatives within the developed neighborhood search. This data structure is referred to as *finite queue*. A finite queue, denoted $\ell^{|n|}$, is a permutation whose size cannot exceed n. Circular lists is an efficient representation of finite queues in computer programming. For a given element e, where $e \notin \ell^{|n|}$, the operation $\ell^{|n|} \swarrow^{[y]} (e)$ results in a new finite queue defined as follows.

$$\ell^{|n|} \swarrow^{[y]} (e) = \begin{cases} \ell^{|n|}(1, y-1) \leftarrow (e) \leftarrow \ell^{|n|}(y, \left|\ell^{|n|}\right|) & \text{if } \left|\ell^{|n|}\right| < n \\ \ell^{|n|}(1, y-1) \leftarrow (e) \leftarrow \ell^{|n|}(y, n-1) & \text{Otherwise} \end{cases} \tag{1}$$

3. Problem definition, notations and network model

The FJSP-SDST has the same structure of the problem in [25] for which an MILP model is presented. There is a set J of jobs ready to be processed in an FMS consisting of a set M of CNC machines which are assumed to be initially idle. Each job requires a predetermined, mandatory sequence of manufacturing operations to be conducted on a subset of M. The index set of operations of job $k \in J$ is I_k= $\{\alpha_k, \alpha_k + 1, ..., \omega_k\}$; where for any two operation indexes i and $j \in I_k$, whenever $i < j$, operation i is to be processed before operation j. The index set for all operations is denoted I= $\bigcup_{k \in J} I_k$= {1,2,...n}. In addition, two dummy operation indexes, 0 and $n + 1$, are used. The job to which operation $i \in I$ belongs is denoted $\beta(i)$.

We denote by $\nabla(i)$ the index of the operation that directly precedes operation i in the job sequence, defined as $\nabla(i) = i - 1$ if $i > \alpha_{\beta(i)}$ and $\nabla(i) = 0$ otherwise. Similarly, $\Delta(i)$ denotes the index of the operation that directly succeeds operation i in the job sequence, defined as $\Delta(i) = i+1$ if $i < \omega_{\beta(i)}$ and $\Delta(i) = n + 1$ otherwise. For each operation $i \in I$, there is a subset of machines $\mu(i) \subseteq M$ that can perform it and from which one machine is to be chosen, and the processing time needed on machine $m \in \mu(i)$ is $p_i^m \in \Re^+$. For the dummy operation 0, we define $p_0^m = 0$ for all $m \in M$; while for the dummy operation $n + 1$ no processing time is defined. For both dummy operations, no machine selection decision is to be made.

The setup time for operation i on machine m when operation j is processed directly before it on the same machine is $s_{j,i}^m \geq 0$. In case there is no operation processed before operation i, the separable setup time is denoted $s_{0,i}^m$. The value $s_{j,i}^m$ is defined only under the conditions that $m \in \mu(i) \cap \mu(j)$, and if $\beta(i) = \beta(j)$ it must be $i = j + 1$. The value $s_{0,i}^m$ is defined only when $m \in \mu(i)$. We assume that the following properties for the setup times hold.

Assumption 1. (Setup time triangular inequality and symmetry) *The setup times between operations follow the triangular inequality property. That is for any three operation indexes $u \in I \cup \{0\}$ and $v, w \in I$ for which setup times with machine index $m \in M$ are defined, the relationship $s_{u,w}^m \leq s_{u,v}^m + s_{v,w}^m$ holds. Furthermore, setup times are symmetric, i.e. $s_{v,w}^m = s_{w,v}^m$ whenever they are both defined.*

Assumption 2. *Sequence-dependent setup times are relatively small compared to the processing times. That is to say* $s^m_{i,j} \ll p^m_j \; \forall i, j, m$ *whenever* $s^m_{i,j}$ *is defined.*

Assumption 1 is justifiable in the case when the sequence-dependent setup time corresponds to the time needed to change tools between operations. Assumption 2 is acceptable in most flexible manufacturing systems as well as manually operated ones. If this assumption is not suitable for some applications, this may limit the applicability of the methodology developed in the rest of this paper.

It is assumed that whenever an operation starts, there will be neither interruption nor preemption. A machine is capable of either performing setup for or processing only one operation at a time. There is no delay due to external factors such that an operation can start its processing on a selected machine as soon as its part becomes available at that machine, the machine is unoccupied by another operation and the separable setup necessary for that part has been completed. Tools and other necessary supporting materials are always available whenever needed by any operation. Tables 1 and 2 present a numerical data for a sample FJSP-SDST instance. In this instance, $J = \{J1, J2, J3\}$ and $M = \{M1, M2, M3\}$.

It is required to select a machine for each operation $i \in I$, denoted $\bar{m}_i$, and to determine its start time, denoted t_i, such that the mandatory processing sequence of operations defined by their corresponding jobs is satisfied along with all aforementioned constraints. The objective is to minimize the maximum completion time–the makespan- defined as $C_{\max} = \max_{k \in J} \left\{ t_{\omega_k} + p^{\bar{m}_{\omega_k}}_{\omega_k} \right\}$.

By extending the permutation-induced acyclic network (PIAN) representation of the JSP [29], the studied problem can be modeled as follows. Each operation is represented by a node having the operation index as its label. A node represents the event of starting the processing of that operation. In addition, There are two nodes representing the events of starting and completing the schedule. These two nodes are respectively labeled 0 and $n + 1$. The set of all nodes is denoted $V = \{\text{node labeled } i\text{: } 0 \leq i \leq n + 1\}$.

The precedence relationships among operations belonging to the same job k are represented by a set of directed arcs called job precedence arcs and defined as $A_k = \{(i, i + 1)\text{: } \alpha_k \leq i < \omega_k\}$. The length of the arc $(i, i+1) \in A_k$ equals the processing time of operation i on its selected machine ($p^{\bar{m}_i}_i$) if a machine selection decision has been made; otherwise, it is set to zero. Furthermore, for each job $k \in J$, arc $(\omega_k, n + 1)$ is defined which has a length of $p^{\bar{m}_{\omega_k}}_{\omega_k}$ if the machine selection decision for operation ω_k has been made and zero otherwise. If $A = \{(\omega_k, n + 1)\} \cup \bigcup_{k \in J} A_k$, the network $N = (V; A)$ represents all jobs' precedence constraints.

The permutation π_m is used to represent the processing sequence of the operations that are selected to be processed on machine $m \in M$, i.e. $\bar{m}_i = m \in \mu(i)$ for every $i \in \pi_m$. Based on a given permutation π_m, the following sets of arcs are defined.

$$z_0(\pi_m) = \{\text{arc } (0, i) \text{ of length } s^m_{0,i} : i \in \pi_m \text{and } \xi(\pi_m, i) = 1\} \tag{2}$$

$$z_1(\pi_m) = \{\text{arc } (i, j) \text{ of length } (p^m_i + s^m_{i,j}) : i, j \in \pi_m \text{and } \xi(\pi_m, j) - \xi(\pi_m, i) = 1\} \tag{3}$$

For every machine $m \in M$, the set of processing sequence arcs $Z(\pi_m) = z_0(\pi_m) \cup z_1(\pi_m)$ represents the processing sequence constraints on that machine.

A solution of the FJSP-SDST can be defined in terms of the vector of permutations $\Pi = (\pi_m : m \in M)$. There are three conditions on Π to represent feasible solutions. These conditions are: (1) $\bigcup_{m \in M} \mathbb{S}(\pi_m) = I$, (2) for every $q, r \in M$, it should be $\mathbb{S}(\pi_q) \cap \mathbb{S}(\pi_r) = \emptyset$, and (3)

the network $\Gamma(\Pi) = (V; A \cup \bigcup_{m \in M} Z(\pi_m))$ must be acyclic. Figure 1 illustrates a solution to the sample instance presented earlier, where the solution is defined as $\Pi = (\pi_{M1}, \pi_{M2}, \pi_{M3}) = ((8 \rightarrow 5 \rightarrow 2), (1 \rightarrow 4), (7 \rightarrow 6 \rightarrow 3))$. In the network representation shown in Figure 1 (a), solid lines represent the set arcs A, while dash lines represent the set of arcs $\bigcup_{m \in M} Z(\pi_m)$.

A path from node i to node j on $\Gamma(\Pi)$, which is a sequence of directed arcs (i,i_1), (i_1, i_2), $\ldots$, (i_q, j), is represented here as a permutation of node (operation) indexes as $(i \rightarrow i_1 \rightarrow i_2 \rightarrow \ldots i_q \rightarrow j)$. A path length is defined as the summation of the directed arcs' lengths in that path. The longest path length from node i to node j on $\Gamma(\Pi)$ is denoted $\ell^{\Pi}(i, j)$. The makespan associated with a permutation vector Π is $C^{\Pi}_{\max} = \ell^{\Pi}(0, n+1)$. The earliest start time of an operation i equals $\ell^{\Pi}(0, i)$ which can be evaluated using the critical path method as in project networks. The method used for assigning operations' start times as $t_i = \ell^{\Pi}(0, i)$ yields semi-active schedules.

It is important to note here that the PIAN representation used for the JSP as well as the disjunctive graph model considers arcs that represent every precedence relationship among all operations processed on the same machine. This would generate additional arcs (i, j) of length p^m_i for all $i, j \in \pi_m$ whenever $\xi(\pi_m, j) - \xi(\pi_m, i) > 1$. In FJSP-SDST, such additional arcs will have lengths calculated as $p^m_i + s^m_{i,j}$ but will have no effect on the determination of the longest path; therefore, they need not to be considered. This point is illustrated by the following proposition. Furthermore, such additional arcs will have no effect on verifying the feasibility condition of having acyclic $\Gamma(\Pi)$ network since there already exists a path from node i to node j.

Proposition 1. *For a given feasible vector of permutations* $\Pi = (\pi_m : m \in M)$ *and nodes* i *and* j *in* $\Gamma(\Pi)$*, where* $i, j \in \pi_m$ *and* $\xi(\pi_m, j) - \xi(\pi_m, i) > 1$ *for some machine* m*, the length of an arc directly connecting* i *and* j *which equals* $p^m_i + s^m_{i,j}$*, will always be less than* $\ell^{\Pi}(i, j)$*.*

Proof. Suppose that the length of the arc (i, j) when $\xi(\pi_m, j) - \xi(\pi_m, i) > 1$ is greater than or equal $\ell^{\Pi}(i, j)$. It follows that for the sub-permutation $\pi' = (i \rightarrow \ldots \rightarrow j) \subset \pi_m$, the summation of the lengths of the processing sequence arcs which equals $\sum_{(q \rightarrow r) \subset \pi'} (p_q + s^m_{q,r})$ is less than $p_i + s^m_{i,j}$, which contradicts assumption 1. □

4. Neighborhood search guiding rules

In this section, we extend the neighborhood search procedure of Mastrolilli and Gambradella [9] developed for the FJSP. We identify some characteristics of a given solution to feasibly and efficiently search the neighborhood.

The head of a given node $i \in V$ is defined as $\eta^{\Pi}_i = \ell^{\Pi}(0, i)$ and its tail $\tau^{\Pi}_i = \ell^{\Pi}(i, n+1)$. The head represents the earliest start time of an operation, while its tail represents the minimum length of time needed after starting an operation before the end of the schedule can be reached. Another parameter that can be derived from the tail is the latest start time of operation i, denoted and defined as $\lambda^{\Pi}_i = C^{\Pi}_{\max} - \tau^{\Pi}_i$.

For a given feasible processing sequences vector Π, there could be more than one longest path from node 0 to node $n+1$ on $\Gamma(\Pi)$. We denote by $\Theta(\Pi)$ the set of such longest paths for which the following property holds.

$$\eta^{\Pi}_i + \tau^{\Pi}_i = \ell^{\Pi}(0, n+1) = C^{\Pi}_{\max} \;\; \forall\, i \in \theta \;\forall\, \theta \in \Theta(\Pi) \tag{4}$$

or alternatively,

$$\eta^{\Pi}_i = \lambda^{\Pi}_i \;\; \forall\, i \in \theta \;\forall\, \theta \in \Theta(\Pi) \tag{5}$$

In the studied problem, a neighborhood function works by moving an operation from its currently assigned position in the current solution Π, which can involve changing the selected machine. Obviously, as in the case of the JSP [30], moving an operation that does not belong to any longest path either keeps the current longest path length unchanged or results in new longest path(s) with larger length. This rule applies to the cases of the FJSP and FJSP-SDST. Therefore, an improvement of a current solution may be achieved by moving an operation $i \in \theta$ in any longest path $\theta \in \Theta(\Pi)$. In the following subsections, we investigate the necessary conditions for both feasibility and improvement of the neighborhood search.

4.1. Maintaining feasibility

For a given feasible solution $\Pi=(\pi_m : m \in M)$, an operation v is to be removed from its selected machine's processing sequence, resulting in a new incomplete processing sequence $\pi_{\bar{m}_v}^{-v} = \pi_{\bar{m}_v} - (v)$ and a vector of processing sequences Π^{-v} containing the same permutations as in Π except that $\pi_{\bar{m}_v}^{-v}$ replaces $\pi_{\bar{m}_v}$. The network $\Gamma(\Pi^{-v})$ is then constructed as described in Sect. 3 with setting the length of the arc $(v, \Delta(v))$ to zero. Accordingly, the arcs (i, v) and $(v, j) \in Z(\pi_{\bar{m}_v})$ are removed from $\Gamma(\Pi)$ and the arc (i, j) is added to generate $\Gamma(\Pi^{-v})$. In case that the arc (v, j) does not exist, i.e. $\xi(\pi_{\bar{m}_v}, v) = \left|\pi_{\bar{m}_v}\right|$, $\Gamma(\Pi^{-v})$ will be generated by just removing the arc $(i, v) \in Z(\pi_{\bar{m}_v})$. Consequently, since the original network $\Gamma(\Pi)$ is acyclic, the network $\Gamma(\Pi^{-v})$ will also be acyclic. Furthermore, for every $i \in V$, we have $\eta_i^{\Pi^{-v}} \leq \eta_i^{\Pi}$, $\tau_i^{\Pi^{-v}} \leq \tau_i^{\Pi}$ and $C_{\max}^{\Pi^{-v}} \leq C_{\max}^{\Pi}$. It can be easily shown that $\tau_{\Delta(v)}^{\Pi} = \tau_{\Delta(v)}^{\Pi^{-v}} = \tau_v^{\Pi^{-v}}$ and $\eta_{\nabla(v)}^{\Pi} = \eta_{\nabla(v)}^{\Pi^{-v}} < \eta_v^{\Pi^{-v}}$ since $\eta_v^{\Pi^{-v}} = \eta_{\nabla(v)}^{\Pi^{-v}} + p_{\nabla(v)}^{\bar{m}_{\nabla(v)}}$.

Operation v is then to be inserted into a processing sequence $\pi_k \in \{\pi_m : \forall m \in \mu(v) \backslash \{\bar{m}_v\}\} \cup \{\pi_{\bar{m}_v}^{-v}\}$. The insertion position in π_k has to be chosen such that the resultant network does not contain cycles. If such condition is satisfied, this process is referred to as *feasible insertion*. For the sake of obtaining feasible insertions, the following two sets of operations are defined.

$$R_{v,k} = \{i : i \in \pi_k \text{ and } \eta_i^{\Pi} + p_i^k > \eta_v^{\Pi^{-v}}\} \tag{6}$$

$$\bar{R}_{v,k} = \mathbb{S}(\pi_k) \backslash R_{v,k} \tag{7}$$

Since $\eta_v^{\Pi^{-v}} = \eta_{\nabla(v)}^{\Pi} + p_{\nabla(v)}^{\bar{m}_{\nabla(v)}}$, set $R_{v,k}$ includes all the operations in π_k that have earliest finish times greater than the earliest start time of $\nabla(v)$ in the feasible solution Π. This assures that there is no path originating at any node that corresponds to an operation in $R_{v,k}$ and ending at node $\nabla(v)$ in the network $\Gamma(\Pi)$. Since there is only one arc coming into node v that starts at node $\nabla(v)$ in the network $\Gamma(\Pi^{-v})$, we are assured that the nodes that correspond to the operations in $R_{v,k}$ do not have any path that ends at node v in the network $\Gamma(\Pi^{-v})$. Meanwhile, for each $i \in \bar{R}_{v,k}$, it can be shown that $\eta_i^{\Pi^{-v}} < \eta_v^{\Pi^{-v}}$. Since $\eta_v^{\Pi^{-v}}$ equals the earliest finish time of operation v as its processing time is temporarily set to zero, we can conclude that there is no path originating at node v and ending at any node $i \in \bar{R}_{v,k}$ in $\Gamma(\Pi^{-v})$. In addition, the following two sets are also defined.

$$L_{v,k} = \{i : i \in \pi_k \text{ and } \tau_i^{\Pi} > \tau_v^{\Pi^{-v}}\} \tag{8}$$

$$\bar{L}_{v,k} = \mathbb{S}(\pi_k) \backslash L_{v,k} \tag{9}$$

Since $\tau_{\Delta(v)}^{\Pi} = \tau_v^{\Pi^{-v}}$, set $L_{v,k}$ includes all the operations in π_k that have tails greater than the tail of $\Delta(v)$ in the feasible solution Π. This assures that in the network $\Gamma(\Pi)$, there is no path originating at node $\Delta(v)$ and ending at a node that corresponds to an operation in set $L_{v,k}$. Since

there is only one arc starting from node v that ends at node $\Delta(v)$ in the network $\Gamma(\Pi^{-v})$, we are assured that the nodes that correspond to the operations in $L_{v,k}$ do not have any path that starts from node v in the network $\Gamma(\Pi^{-v})$. Meanwhile, for each $i \in \bar{L}_{v,k}$, we have $\tau_i^{\Pi^{-v}} \leq \tau_v^{\Pi^{-v}}$ which can be split into two cases. In the first case we have $\tau_i^{\Pi^{-v}} < \tau_v^{\Pi^{-v}}$ which means that there is no path originating at node $i \in \bar{L}_{v,k}$ and ending at node v. In the second case we have $\tau_i^{\Pi^{-v}} = \tau_v^{\Pi^{-v}}$. Since $\tau_v^{\Pi^{-v}} = \tau_{\Delta(v)}^{\Pi^{-v}}$ and there is only one arc connecting node v to $\Delta(v)$, either we have $i = \Delta(v)$ in which case we know that there is no path from i to v in $\Gamma(\Pi^{-v})$ or $i \neq \Delta(v)$. It can be easily shown that the latter case in which $\tau_i^{\Pi^{-v}} = \tau_v^{\Pi^{-v}}$ and $i \neq \Delta(v)$ cannot actually happen unless there is no path originating at node i and ending at node v. We can conclude that there is no path originating at any node $i \in \bar{L}_{v,k}$ and ending at node v in $\Gamma(\Pi^{-v})$.

Given the four sets defined by equations 6 through 9, the following two positions in π_k are defined.

$$\underline{x}_{v,k} = \begin{cases} \min\{\xi(\pi_k, i) : i \in R_{v,k} \cup \bar{L}_{v,k}\} - 1 & \text{if } R_{v,k} \cup \bar{L}_{v,k} \neq \emptyset \\ |\pi_k| & \text{otherwise} \end{cases} \tag{10}$$

$$\bar{x}_{v,k} = \begin{cases} \max\{\xi(\pi_k, i) : i \in L_{v,k} \cup \bar{R}_{v,k}\} & \text{if } L_{v,k} \cup \bar{R}_{v,k} \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

The following proposition states a sufficient condition on the value of the insertion position of operation v in π_k for obtaining guaranteed feasible insertions.

Proposition 2. *A feasible insertion of operation v into π_k is guaranteed if the insertion position x is selected such that $\underline{x}_{v,k} \leq x \leq \bar{x}_{v,k}$.*

4.2. Rules for improvement

By inserting operation v into π_k at position x, the resulting permutation is denoted π_k^{+v}. As described in Sect. 2, this operation is represented as $\pi_k^{+v}{=}\pi \xleftarrow{x} (v)$. The resulting vector of complete processing sequences is denoted $\Pi_{v,k}$ which contains the same permutations as Π^{-v} except that π_k^{+v} replaces π_k. If the insertion position is chosen according the condition given in proposition 2, the network $\Gamma(\Pi_{v,k})$ is guaranteed to be acyclic. The resulting makespan, denoted $C_{\max}^{\Pi_{v,k}}$, has the following property.

Proposition 3. $C_{\max}^{\Pi_{v,k}} \geq C_{\max}^{\Pi^{-v}}$.

Proof. As a result of the insertion process, we have the following possible changes to generate the network $\Gamma(\Pi_{v,k})$ from $\Gamma(\Pi^{-v})$ depending on the value of the insertion position x.

Case 1: When $0 < x < |\pi_k|$, the processing sequence arc $(\pi_k[x], \pi_k[x+1])$ is removed and the two processing sequence arcs $(\pi_k[x], v)$ and $(v, \pi_k[x+1])$ are added.

Case 2: When $x = 0$, the processing sequence arc $(0, \pi_k[x])$ with length $= s_{0,\pi_k[x]}$ according to equation (2) will be removed and the two arcs $(0, v)$ and $(v, \pi_k[1])$ will be added.

Case 3: When $x = |\pi_k|$, there will be only one arc $(\pi_k[x], v)$ added with a length given in equation (3) and no arcs will be removed.

According to the lengths of the processing sequence arcs given in equations (2) and (3) along with assumption 1 regarding the triangular inequality of sequence-dependent setup times, we can conclude that for all the above three cases it should be $\eta_i^{\Pi^{-v}} \leq \eta_i^{\Pi_{v,k}}$ and $\tau_i^{\Pi^{-v}} \leq \tau_i^{\Pi_{v,k}}$ for all $i \in V$. Consequently $C_{\max}^{\Pi^{-v}} \leq C_{\max}^{\Pi_{v,k}}$. □

Accordingly, to improve a current solution, operation v has to belong to a critical path as stated earlier for otherwise $C_{\max}^{\Pi} = C_{\max}^{\Pi^{-v}}$ and there will be no improvement. In addition, the lower the value of $C_{\max}^{\Pi^{-v}}$ is, the higher the potential will be for improving the current solution. If the insertion process results in $C_{\max}^{\Pi_{v,k}} > C_{\max}^{\Pi^{-v}}$, this means that operation v must be included in a critical path in $\Gamma(\Pi_{v,k})$ and accordingly $\eta_v^{\Pi_{v,k}} + \tau_v^{\Pi_{v,k}} = C_{\max}^{\Pi_{v,k}}$. The following equation represents this relationship.

$$C_{\max}^{\Pi_{v,k}} = \max\{C_{\max}^{\Pi^{-v}}, \eta_v^{\Pi_{v,k}} + \tau_v^{\Pi_{v,k}}\} \tag{12}$$

Let $u(x) = \pi_k[x]$ when $x > 0$, and $u(x) = 0$ when $x = 0$. It can be easily shown that $\eta_v^{\Pi_{v,k}} = \max\{\eta_v^{\Pi^{-v}}, \eta_{u(x)}^{\Pi^{-v}} + s_{u(x),v}^k + p_{u(x)}^k\}$. Similarly, $\tau_v^{\Pi_{v,k}} = \max\{\tau_v^{\Pi^{-v}} + p_v^k, \tau_{w(x)}^{\Pi^{-v}} + s_{v,w(x)}^k + p_v^k\}$ where $w(x) = \pi_k[x+1]$ for $x < |\pi_k|$. When $x = |\pi_k|$, $\tau_v^{\Pi_{v,k}} = \tau_v^{\Pi^{-v}} + p_v^k$. Accordingly we have the following relationship.

$$\begin{aligned}\eta_v^{\Pi_{v,k}} + \tau_v^{\Pi_{v,k}} &= \begin{cases} \max\{\eta_v^{\Pi^{-v}}, \eta_{u(x)}^{\Pi^{-v}} + s_{u(x),v}^k + p_{u(x)}^k\} & \\ \quad + \max\{\tau_v^{\Pi^{-v}} + p_v^k, \tau_{w(x)}^{\Pi^{-v}} + s_{v,w(x)}^k + p_v^k\} & x < |\pi_k| \\ \max\{\eta_v^{\Pi^{-v}}, \eta_{u(x)}^{\Pi^{-v}} + s_{u(x),v}^k + p_{u(x)}^k\} + \tau_v^{\Pi^{-v}} + p_v^k & x = |\pi_k| \end{cases} \\ &= (\eta_v^{\Pi^{-v}} + \tau_v^{\Pi^{-v}}) + p_v^k + \delta_v(x) \end{aligned} \tag{13}$$

where,

$$\delta_v(x) = \max\{0, \delta_v^1(x)\} + \begin{cases} \max\{0, \delta_v^2(x)\} & x < |\pi_k| \\ 0 & x = |\pi_k| \end{cases} \tag{14}$$

$$\delta_v^1(x) = \eta_{u(x)}^{\Pi^{-v}} + s_{u(x),v}^k + p_{u(x)}^k - \eta_v^{\Pi^{-v}} \tag{15}$$

$$\delta_v^2(x) = \tau_{w(x)}^{\Pi^{-v}} + s_{v,w(x)}^k - \tau_v^{\Pi^{-v}} \tag{16}$$

Consequently, for a given selection of operation v, the determination of machine k and the value of the insertion position x can be done by seeking the minimum of the value of $\eta_v^{\Pi_{v,k}} + \tau_v^{\Pi_{v,k}}$ which reduces to the minimization of $\delta_v(x)$ given in equation (14). The following lemmas and theorem state important properties based upon which the best insertion position can be determined.

Lemma 1. *For any two insertion positions x_1 and x_2 of operation v in π_k where $x_2 > x_1$, it should be $\delta_v^1(x_2) > \delta_v^1(x_1)$.*

Proof. In the case when $x_1 > 0$, consider the sub-permutation $(a \rightarrow b \rightarrow c) \subset \pi_k$, and let $\pi_k[x_1] = a$ and $\pi_k[x_2] = b$, i.e. the first insertion position is chosen between a and b, and the second insertion position is chosen between b and c. Here, $\delta_v^1(x_1) = \eta_a^{\Pi^{-v}} + s_{a,v}^k + p_a^k - \eta_v^{\Pi^{-v}}$ and $\delta_v^1(x_2) = \eta_b^{\Pi^{-v}} + s_{b,v}^k + p_b^k - \eta_v^{\Pi^{-v}}$. Since $\eta_b^{\Pi^{-v}} - \eta_a^{\Pi^{-v}} \geq p_a^k + s_{a,b}^k$ and from assumption 1 we know that $s_{a,b}^k + s_{b,v}^k \geq s_{a,v}^k$ which means that $s_{a,b}^k + s_{b,v}^k + p_b^k > s_{a,v}^k$, we conclude that $\delta_v^1(x_2) > \delta_v^1(x_1)$. In the case when $x_1 = 0$ similar analysis will lead to the same conclusion. This result can be generalized to all insertion positions satisfying the hypothesis. □

Lemma 2. *For any two insertion positions x_1 and x_2 of operation v in π_k where $x_2 > x_1$ and $x_2 < |\pi_k|$, it should be $\delta_v^2(x_2) < \delta_v^2(x_1)$.*

Proof. Consider the sub-permutation $(a \rightarrow b \rightarrow c) \subset \pi_k$, and let $\pi_k[x_1] = a$ and $\pi_k[x_2] = b$. Here, $\delta_v^2(x_1) = \tau_b^{\Pi^{-v}} + s_{v,b}^k - \tau_v^{\Pi^{-v}}$ and $\delta_v^2(x_2) = \tau_c^{\Pi^{-v}} + s_{v,c}^k - \tau_v^{\Pi^{-v}}$. Since $\tau_b^{\Pi^{-v}} - \tau_c^{\Pi^{-v}} \geq p_b^k + s_{b,c}^k$ and from assumption 1 we know that $s_{v,b}^k + s_{b,c}^k \geq s_{v,c}^k$ which means $s_{v,b}^k + s_{b,c}^k + p_b^k > s_{v,c}^k$, we conclude that $\delta_v^2(x_2) < \delta_v^2(x_1)$. This result can be generalized to all insertion positions satisfying the hypothesis. □

Theorem 1. *There exists an insertion position $x_{v,k}^*$ where $\underline{x}_{v,k} \leq x_{v,k}^* \leq \bar{x}_{v,k}$ for which $\delta_v(x_{v,k}^*)$ is minimum.*

As a result of theorem 1, the search space for the insertion position will be restricted to the domain given by $\underline{x}_{v,k} \leq x \leq \bar{x}_{v,k}$ which at the same time guarantees that the resultant solution will be feasible as stated by proposition 2.

However, to accurately determine the best insertion position within the specified range, the value of $\delta_v(x)$ needs to be evaluated at each position in the range for every operation-machine combination (v, k). This process is in fact computationally expensive since it involves the calculation of the heads and tails of all the operations in the modified network $\Gamma(\Pi^{-v})$. Instead, as proposed by Mastrolilli and Gambradella [9] for the FJSP, an upper bound on $\delta_v(x)$ can replace its exact value as an approximation. Such an upper bound, denoted $\bar{\delta}_v(x)$, should be based on information available in the network $\Gamma(\Pi)$ so that no need to construct the network $\Gamma(\Pi^{-v})$. $\bar{\delta}_v(x)$ is calculated by replacing $\delta_v^1(x)$ and $\delta_v^2(x)$ defined in equations (15) and (16) with $\bar{\delta}_v^1(x)$ and $\bar{\delta}_v^2(x)$ respectively, which are evaluated as follows.

$$\bar{\delta}_v^1(x) = \eta_{u(x)}^{\Pi} + s_{u(x),v}^k + p_{u(x)}^k - (\eta_{\nabla(v)}^{\Pi} + p_{\nabla(v)}^{\bar{m}_{\nabla(v)}}) \tag{17}$$

$$\bar{\delta}_v^2(x) = \tau_{w(x)}^{\Pi} + s_{v,w(x)}^k - \tau_{\Delta(v)}^{\Pi} \tag{18}$$

5. Neighborhood search function

Based on the guidelines outlined in Sect. 4, a simple greedy neighborhood search function (GNS) is firstly described here. For a given incumbent solution Π, the GNS makes a decision on the local move that will be conducted to obtain a new neighborhood solution. A local move is defined using the tuple (v, k, x), where v is the selected operation to be moved, k is the machine to which the selected operation will be moved and x is the insertion position in π_k. The upper bound associated with a move is defined as follows.

$$\mathrm{UB}(v, k, x) = \eta_v^{\Pi^{-v}} + \tau_v^{\Pi^{-v}} + p_v^k + \bar{\delta}_v(x). \tag{19}$$

The selection of a move in the GNS function is done sequentially by deciding first on the operation to be moved, then the machine to which the selected operation will be moved, and finally the insertion position. Each step of that sequence of decisions is done in a greedy manner that exploits the structure that defines the upper bound. An operation that belongs to a critical path and has the minimum value of $\eta_i^{\Pi^{-i}} + \tau_i^{\Pi^{-i}}$ is firstly selected and labeled v. Then the machine that has the minimum processing time for the selected operation v is selected and denoted k. All feasible insertion positions that belong to the range from $\underline{x}_{v,k}$ to $\bar{x}_{v,k}$ (excluding the current position if $k = \bar{m}_v$) are examined to find the best insertion position with the minimum $\bar{\delta}_v(x)$ value. If a move is found to achieve an upper bound that is less than the best found makespan, the GNS function selects it and updates the current solution accordingly. Otherwise, the search

continues for all other operation-machine-insertion postions combinations with the same greedy selection rules until the best move with the minimum upper bound is found. In case that for all critical operations no single move is found after investigating all possible machines and all possible optimal ranges of insertion positions, the GNS terminates.

Strictly applying the GNS function will lead to a sequence of cyclic moves and consequently entrapment in a local minimum. We introduce here a mechanism in the neighborhood search for randomizing some decisions as a way of escaping local minima. The resultant neighborhood search function is referred to as randomized greedy neighborhood search (RGNS).

In the RGNS function, a list of the best-found local moves that are associated with the critical operations, denoted $\ell_M^{|n_{lm}|}$, is maintained using the finite queue structure. Similarly, a list of the best found insertion positions for each critical operation is maintained for each candidate machine. This list is denoted $\ell_P^{|n_{Pos}|}$. The sizes of both lists, n_{lm} and n_{Pos}, are passed on to the RGNS by the metaheuristic using it depending on the level of randomization needed. The RGNS function also receives the best makespan found, $C^*_{\max}$. The following list describes the steps of the RGNS function.

Function RGNS(Receives Π ,$C^*_{\max}$, n_{lm} and n_{Pos}) : returns new makespan

1. Let $\ell_M^{|n_{lm}|} = (.)$
2. Let set Ψ contain all nodes (operations) belonging to the critical paths in the network $\Gamma(\Pi)$
3. For every operation $i \in \Psi$ determine $\eta_i^{\Pi^{-i}} = \eta^{\Pi}_{\nabla(i)} + p^{\bar{m}_{\nabla(i)}}_{\nabla(i)}$ and $\tau_i^{\Pi^{-i}} = \tau^{\Pi}_{\Delta(i)}$
4. Select operation $v = \arg\min_{i\in\Psi} \left\{\eta_i^{\Pi^{-i}} + \tau_i^{\Pi^{-i}}\right\}$ with arbitrary tie-breaking
5. Let $\pi_{\bar{m}_v}$ denote the current processing sequence that includes operation v where $\bar{m}_v$ denotes its currently assigned machine in Π
6. Let set $\Omega = \mu(v)$
7. Select machine $k = \arg\min_{m\in\Omega} \{p_v^m\}$
8. Determine the lower and upper limits of the insertion position, $\underline{x}_{v,k}$ and $\bar{x}_{v,k}$, as defined by equations (6) through (11)
9. Let $\ell_P^{|n_{Pos}|} = (.)$
10. For every $x \in \{\underline{x}_{v,k}, \ldots \bar{x}_{v,k}\}\backslash\{x' : x' = \xi(\pi_{\bar{m}_v}, v) - 1 \text{ whenever } k = \bar{m}_v\}$, let set $P = \{z : z \in \ell_P^{|n_{Pos}|} \text{ and } \bar{\delta}_v(z) > \bar{\delta}_v(x)\}$.
11. If $P = \emptyset$ and $\left|\ell_P^{|n_{Pos}|}\right| < n_{Pos}$ then let $\ell_P^{|n_{Pos}|} = \ell_P^{|n_{Pos}|} \leftarrow (x)$; else if $P \neq \emptyset$ then let $y = \xi(\ell_P^{|n_{Pos}|}, \arg\min_{z\in P}(\bar{\delta}_v(z)))$, and $\ell_P^{|n_{Pos}|} = \ell_P^{|n_{Pos}|} \swarrow^{[y]} (x)$
12. If $\ell_P^{|n_{Pos}|} = (.)$ then go to step 17
13. Select $x^*_{v,k}$ from $\ell_P^{|n_{Pos}|}$ randomly
14. If $\mathrm{UB}(v, k, x^*_{v,k}) < C^*_{\max}$ then let $\ell_M^{|n_{lm}|} = \left((v, k, x^*_{v,k})\right)$ and go to step 20
15. Let set $\mathfrak{M} = \{(\bar{v}, \bar{k}, \bar{x}) : (\bar{v}, \bar{k}, \bar{x}) \in \ell_M^{|n_{lm}|} \text{ and } \mathrm{UB}(\bar{v}, \bar{k}, \bar{x}) > \mathrm{UB}(v, k, x^*_{v,k})\}$.
16. If $\mathfrak{M} = \emptyset$ and $\left|\ell_M^{|n_{lm}|}\right| < n_{lm}$ then let $\ell_M^{|n_{lm}|} = \ell_M^{|n_{lm}|} \leftarrow \left((v, k, x^*_{v,k})\right)$; else if $\mathfrak{M} \neq \emptyset$ then let $y = \xi(\ell_M^{|n_{lm}|}, \arg\min_{(\bar{v},\bar{k},\bar{x})\in\mathfrak{M}}(\mathrm{UB}(\bar{v}, \bar{k}, \bar{x}))$, and $\ell_M^{|n_{lm}|} = \ell_M^{|n_{lm}|} \swarrow^{[y]} \left((v, k, x^*_{v,k})\right)$
17. Let $\Omega = \Omega\backslash\{k\}$; if $\Omega \neq \emptyset$ then go to step 7
18. Let $\Psi = \Psi\backslash\{v\}$; if $\Psi \neq \emptyset$ then go to step 4

19. If $\ell_M^{|n_{lm}|} = (.)$ then return $C_{\max}^{\Pi}$
20. Randomly select a move $(\hat{v}, \hat{k}, \hat{x})$ from $\ell_M^{|n_{lm}|}$
21. Modify the current vector of processing sequences, Π, by conducting the two operations $\pi_{\bar{m}_{\hat{v}}} = \pi_{\bar{m}_{\hat{v}}} - (\hat{v})$ and $\pi_{\hat{k}} = \pi_{\hat{k}} \overset{\hat{x}}{\longleftarrow} (\hat{v})$, then evaluate and return the new makespan $C_{\max}^{\Pi}$

The RGNS function starts in step 1 with an empty list of local moves, denoted $\ell_M^{|n_{lm}|}$. In steps 2 and 3, all critical operations are identified and stored in set Ψ, and their corresponding values of $\eta_i^{\Pi^{-i}}$ and $\tau_i^{\Pi^{-i}}$ are evaluated. The main loop starts at step 4 by selecting an operation v from set Ψ that has the minimum value of $\eta_i^{\Pi^{-i}} + \tau_i^{\Pi^{-i}}$. The set of machines that can process operation v is stored in the temporary set Ω in step 6 to be used within the machine selection loop which starts at step 7.

The function starts to investigate all possible machines to which operation v can be moved in step 7 by selecting the machine with the minimum processing time among those machines contained currently in set Ω. For a currently investigated machine k, the lower and upper limits of insertion positions are determined in step 8. Steps 9, 10 and 11 construct the finite queue of best insertion positions, denoted $\ell_P^{|n_{Pos}|}$, within the determined limits excluding the current position of operation v if the investigated machine is the same as the current one in the starting solution. The operations shown in steps 10 and 11 aim to maintain a list of the best insertion positions in $\ell_P^{|n_{Pos}|}$ without exceeding the limit on its size (n_{Pos}). If no such insertion position is available, the function decides in step 12 to investigate another machine.

In step 13, an insertion position is selected randomly from $\ell_P^{|n_{Pos}|}$. If the selected position results in an upper bound that is less than the current best makespan, the search loops are terminated and the current move is directly applied as stated in step 14. Otherwise, the finite queue of moves, denoted $\ell_M^{|n_{lm}|}$, is updated in steps 15 and 16 such that it contains a list of the best moves with the best upper bounds. In steps 17 and 18, the two sets Ω and Ψ are updated respectively and the terminating conditions of the two loops are checked. In step 19, the situation in which no local moves are found is checked. In such a case, the function terminates and returns the initial solution's makespan without modifying it. Otherwise, a move is randomly selected in step 20 and applied to the current solution in step 21, and the function returns the makespan of the modified soluton.

6. Experimentation and results for the neighborhood search function

The purpose of the experimental work in this section is to assess the performance of the RGNS at different levels of both n_{lm} and n_{Pos} under different problem settings. For that sake, a simple hill climbing algorithm is used which is presented in the following list.

HC-RGNS(receives Π_0, $itr^{\max}$, n_{lm} and n_{Pos}) : returns C^*_{max} and itr^*

1. Let $\Pi = \Pi_0$, $C^*_{\max} = C^{\Pi_0}_{\max}$, $itr^* = itr = 1$
2. Let C = RGNS (Π, $C^*_{\max}$, n_{lm}, n_{Pos})
3. If $C < C^*_{\max}$ then let $C^*_{\max} = C$ and $itr^* = itr$
4. If $itr = itr^{\max}$ then return C^*_{max} and itr^*; otherwise let $itr = itr + 1$ and go to step 2

The HC-RGNS algorithm is a simple iterative approach that receives an initial solution (Π_0), maximum number of iterations ($itr^{\max}$) and the two parameters n_{lm} and n_{Pos}. The HC-RGNS algorithm calls the RGNS function at each iteration, passing to it the current solution (Π), the

best makespan found ($C^*_{\max}$) and the two parameters n_{lm} and n_{Pos}. The last two parameters remain fixed throughout the algorithm, while the current solution Π is modified internally within the RGNS function. The makespan value returned from the RGNS function is compared to the best found makespan in step 3, and the values of the best makespan and the iteration number at which the best solution is found (itr^*) are updated. The algorithm terminates when the maximum number of iterations ($itr^{\max}$) is reached, and returns both C^*_{max} and itr^*.

Based on the HC-RGNS algorithm, experimentations are conducted. In the following subsections, the experimental design and results are presented.

6.1. Experimental design

Since the FJSP-SDS is an extension to the FJSP, it is imperative to extend the standard benchmark instances developed in the literature for the FJSP in order to integrate their ineherent properties into the current experimental design. In the literature of FJSP, there are seven sets of standard benchmark instances, summing up to a total of 183 instances. Table 3 lists the references in which each set is defined, the problem set acronym used herein, the number of instances in each set and the ranges of the problems' parameters in each set. Among those parameters is the flexibility ratio (*flex.*) which is defined as the average number of alternate machines per operation approximated here to two digits. When this ratio equals 1.0, the FJSP instance is exactly a traditional JSP case; and the higher this ratio is, the more routing flexibility is represented in the problem.

In the current experimental design, some FJSP instances are selected to be modified by adding sequence-dependent setup times. The selection is done by defining two levels for the flexiblity ratio. The low flexiblity ratio level is defined to be greater than 1.0 and less than 1.96, while the high flexbility ratio level is defined to be greater than or equal to 1.96 and less than or equal to 10.0. The limites used in defining the levels of the flexibility ratio are selected such that both levels contain almost the same number of FJSP instances found in the literature. Here, there are 89 instances in the low flexbility set and 94 instances in the high flexbility set. At each level of the flexbility ratio, 30 problem instances are selected aribtrarily such that the flexiblity ratios are uniformly distributed along the range as much as possible. Tables 4 and 5 list the selected 30 instances for each level of the flexibility ratio.

For each FJSP instance, four settings are defined by specifying two levels for two parameters that are related to the values of the setup times. These two parameters are defined to consider situations in which both assumptions 1 and 2 do and do not hold. The first parameter is related to assumption 1 with two settings defined. The first setting enforces the triangular inequality to hold among all setup time values; while in the second setting, setup times are randomly generated without enforcing triangular inequality. The second parameter is related to assumption 2 with two settings defined. In the first setting the smallest processing time among all operations processed on a given machine is determined, and setup times are randomly generated using uniform distribution within the range from 0.05 to 0.3 of the smallest processing time of each machine. In the second setting, setup times are randomly generated using uniform distribution within the range from the smallest to the largest processing times on a given machine.

In the current experimental design, the triangular inequality property is enforced by defining two coordinate values for each operation. The two coordinates are randomly generated along the specified range of the setup time values divided by $\sqrt{2}$. Then the sequence dependent setup times between any two operations is determined by taking the square root of the summation of the cubic difference between the two coordinates of the two operations and then adding to that the low setup time value of the defined setup time range. The result is then rounded to the nearest

lowest integer value. The initial setup time value ($s^m_{0,i}$) is calculated as the square root of the cubic values of the coordinates generated for operation i. This approach is necessary to make sure that the randomly generated setup time values will remain within the specified range and the triangular inequality is maintained. However, for some instances with small setup times, a post correction algorithm is used to adjust setup times as necessary to make sure that the triangular inequality property is satisfied.

As indicated earlier, the purpose of the experimentation in this section is to assess the performance of the RGNS at different levels of the two parameters n_{lm} and n_{Pos}. For each parameter, five levels are considered, they are: 1, 2, 5, 10 and 20. Therefore, the current experimental design is a general factorial design, where three factors, namely the flexibility ratio ($flex.$), the triangular inequality satisfaction and the relative value of setup times to processing times (s/p ratio), have two settings each; and the values of n_{lm} and n_{Pos} have five levels each. At each combination of settings of those factors, 30 test instances are used in evaluating the neighborhood search function, and HC-RGNS is run for 1000 iterations with 100 runs for each problem instance.

In each run, the initial solution is randomly constructed by using a simple active schedule construction heuristic based on [31] with random rules for machine selection and sequencing of operations. For each run, the percentage improvement ($imp\%$) defined as the difference between the initial makespan and the best makespan, divided by the initial makespan and multiplied by 100 is evaluated. In addition, the iteration number at which the best makespan is achieved (itr^*) is recorded.

6.2. Results and analysis

The performance of the RGNS function is assessed using two measures: the average percentage improvement ($\overline{imp\%}$) and the average best makespan iteration number ($\overline{itr^*}$). These measures are calculated for the 100 runs conducted on the same instance with a specific combination of the considered factors, where each run starts with a different randomly generated initial solution as stated in the previous section.

The average percentage improvement ($\overline{imp\%}$) is a measure of the improvement capability of the RGNS function with a given combination of (n_{lm}, n_{Pos}). It also provides an indication of the relative quality of the final solutions. Therefore, higher $\overline{imp\%}$ values are favorable. The average number of iterations conducted until the best makespan is reached ($\overline{itr^*}$) is a measure of the capability of the RGNS function with a given combination (n_{lm}, n_{Pos}) in escaping local minima. low $\overline{itr^*}$ values indicate that the function is trapped early in a local minimum and no further improvements are found until the iteration counter reached the limit of 1000.

To present the effect of the interaction between the main RGNS parameters, namely (n_{lm} and n_{Pos}), on $\overline{imp\%}$ and $\overline{itr^*}$, the interaction plots are drawn in figures 2 and 3. Each point in the interaction plot represents the average performance of the RGNS function at a specific combination of the two factors. It is evident from the figures that both performance measures are in coherent response with the RGNS parameters. Higher values of n_{Pos} deteriorates the performance at high values of n_{lm}. Meanwhile, at lower values of n_{lm}, specifically 1 and 2, the performance does not improve significantly with the increase of n_{Pos}. It can be concluded from figures 2 and 3 that the best performance in terms of the average percentage improvement can be obtained using the combination of (n_{lm}, n_{Pos}) with the values of (10, 1) and (10, 2). Furthermore, the combination (5, 2) provide a very close performance level.

7. Tabu search approach

In this section, a tabu search approach is presented, which is referred to as randomized-greedy tabu search (RGTS). The RGTS is based on the RGNS function with a minor modification that allows it to receive a tabu list as an input and to prevent tabu moves from being reselected. The modified RGNS is referred to as Tabued RGNS or TRGNS. The tabu list is represented as a finite queue denoted $\ell_T^{|n_{ts}|}$. The following list shows how the TRGNS is modified from the RGNS.

Function TRGNS(Receives Π ,$C^*_{\max}$, n_{lm}, n_{Pos} and $\ell_T^{|n_{ts}|}$) : returns new makespan

1. to 9. Same as in RGNS.
10. For every $x \in \{\underline{x}_{v,k}, \ldots \bar{x}_{v,k}\} \backslash \{x' : x' = \xi(\pi_{\bar{m}_v}, v) - 1 \text{ whenever } k = \bar{m}_v$, and $(v, k, x) \notin \ell_T^{|n_{ts}|}$ unless $\mathrm{UB}(v, k, x) < C^*_{\max}\}$; let set $P = \{z : z \in \ell_P^{|n_{Pos}|} \text{ and } \bar{\delta}_v(z) > \bar{\delta}_v(x)\}$.
11. to 19. Same as in RGNS.
20. Randomly select a move $(\hat{v}, \hat{k}, \hat{x})$ from $\ell_M^{|n_{lm}|}$ and let $\ell_T^{|n_{ts}|} = \ell_T^{|n_{ts}|} \swarrow^{[1]} (\hat{v}, \hat{k}, \hat{x})$
21. Same as RGNS

The modification made in step 10 excludes a tabued move from being reselected unless its upper bound is found to be less than the best found makespan. In step 20, a modification is made to add the selected move to the tabu list. The following list describes the main steps of the developed RGTS procedure.

RGTS(receives n_{ts}, $itr^{\max}$ and LB) : returns best found solution

1. Construct an initial solution Π and let $\ell_T^{|n_{ts}|} = (.)$, $\Pi^* = \Pi$ and $itr = 0$
2. Let C = RGNS (Π, $C^{\Pi^*}_{\max}$, 10, 1, $\ell_T^{|n_{ts}|}$)
3. If $C < C^{\Pi^*}_{\max}$ then let $\Pi^* = \Pi$ and $\ell_T^{|n_{ts}|} = (.)$
4. If $itr = itr^{\max}$ or $C^{\Pi^*}_{\max} = LB$ then return Π^*; otherwise let $itr = itr + 1$ and go to step 2

The RGTS procedure receives the maximum tabu size (n_{ts}), the limit on the number of iterations ($itr^{\max}$) and a lower bound on the makespan (LB). It starts by constructing an initial solution using a simple active schedule generator based on [31] with random dispatching rule for sequencing outstanding operations on a given machine and a modification for machine selection decisions. The procedure iteratively applies the RGNS function with $n_{lm} = 10$ and $n_{Pos} = 1$ to update a current solution Π, and terminates when the number of iterations reaches a specified limit $itr^{\max}$ or when a lower bound is hit. In each iteration, if a better solution is reached the tabu list will be cleared as shown in step 2.

Lower bounds are used for stopping iterations as well as in assessing the quality of the final solutions obtained by evaluating the optimality gap, which is defined as $gap = (C^{\Pi^*}_{\max} - LB)/LB$. The value of LB for a given FJSP-SDST instance is evaluated as follows.

$$LB_1 = \min_{j \in J} \left\{ \sum_{i \in I_j} \min_{m \in \mu(i)} \{p_i^m\} + \min_{\substack{m \in \mu(\alpha_j),\, i \in I \backslash I_j \\ \text{where } m \in \mu(i)}} \left\{ s^m_{i,\alpha_j} \right\} \right\} \tag{20}$$

$$LB_2 = LB^{(\mathrm{FJSP})} + \max_{m \in M} \left\{ \min_{\substack{i \in I \text{ where } m \in \mu(i) \\ \text{given that } \Omega_m \neq \emptyset}} \left\{ s^m_{0,i} \right\} + LB^{(\mathrm{TSP}_{\Omega_m})} \right\} \tag{21}$$

$$LB = \max\{LB_1, LB_2\} \tag{22}$$

In eqation (20), the first lower bound (LB_1) is evaluated by relaxing the machine availability constraint which entails that a machine can not process more than one operation at a time, and assigning each operation to the machine that provides the lowest processing time. By summing up the processing times of the operations and adding to that the minimum possible setup time of the first operation of each job, the minimum of the resultant values among all jobs is provably a lower bound to the FJSP-SDST problem. In Eq. (21), a second lower bound (LB_2) is evaluated based on the lower bounds available for the FJSP [32]. Here, a subset of operations $\Omega_m = \{i : i \in I,\ m \in \mu(i) \text{ and } |\mu(i)| = 1\}$ for which machine m is the only processor is defined. LB_2 equals the summation of the lower bound of the base FJSP instance, denoted $LB^{(\text{FJSP})}$, and the maximum value among machines for the summation of the minimum possible initial setup time and the lower bound of a travelling salesman problem (TSP) defined using the sequence-dependent setup times between operations in set Ω_m as the distances between cities. Here, a relaxation of the TSP as an assignment problem is used to evaluate its lower bound. The proof that LB_2 is a valid lower bound for the FJSP-SDST is straightforward.

Experimentations are conducted to assess the performance of the RGTS approach using the instances generated earlier in section 6.1. For the current RGTS implementation, the tabu size n_{ts} is selected to equal the number of operations n, and the maximum number of iterations $itr^{\max}$ is fixed at 100000. Even though the initial solutions are randomly generated, it is found that the variability in the final makespan values is very limited in most of the cases. Therefore, two runs are conducted for each instance, and the best makespans are reported in tables 6 and 7 along with the computed lower bounds.

Since the quality of the evaluated lower bounds has a direct effect on the optimality gaps which are used to assess the performance of the RGTS, it is important to investigate first the effect of the problem parameter settings on their values. For that sake, the ratio $\Delta(LB) = (LB - LB^{(\text{FJSP})})/LB^{(\text{FJSP})}$ is evaluated for each instance. It is found that at 99% confidence level, both the flexibility ratio and the s/p ratio have significant effect on $\Delta(LB)$, while the triangular inequality has no effect. The ratio $\Delta(LB)$ decreases at higher flexibility ratios which is attributed to the lower values of $LB^{(\text{TSP}_{\Omega_m})}$ as a result of lower cardinality of the set Ω_m. At higher values of the s/p ratio, the ratio $\Delta(LB)$ naturally increases due to the higher values of setup times.

For the optimality gap of the RGTS, it is found that both the flexibility ratio level and the triangular inequality condition do not have significant effects on the optimality gap which is almost level at approximately 25.5% for all instances. Meanwhile, it is found that the optimality gap is significantly affected by the s/p ratios. The average optimality gap is found to be below 10% at low s/p ratios, while it reaches 42% at high s/p ratios. These results indicate that even though the quality of the lower bounds is lower at higher flexiblity ratios, the tabu search approach is capable of reducing the optimality gap for those problems to the point that makes its performance equivelant in both levels of the flexiblity ratio. On the other hand, even though the lower bounds generated for instances with higher s/p ratios are relatively better than those with lower s/p ratios, the optimality gaps for instances with higher s/p ratios are relatively high. This indicates that the application domain of the developed tabu search appraoch is suitable to problems with low s/p ratios, while both flexiblity level and triangular inequality condition do not have effect on its performance.

8. Conclusion and future work

In this paper, the problem of flexible job shop scheduling with separable sequence-dependent setup times is considered with the objective of minimizing the makespan. It is shown that the assumptions of small setup times compared to processing times and the triangular inequality of setup times between operations are necessary to be able to theoretically extend previously developed neighborhood search functions for the flexible job shop scheduling problem. Therefore, a randomized greedy neighborhood search function is introduced. This function utilizes a greedy approach that terminates the search whenever a move is found with an upper bound greater than the best found makespan during the search. Randomization is introduced at two points of the search as a mechanism that helps escaping local minima. This randomization is controlled by two parameters that define the sizes of the sets from which best insertion positions and best moves are randomly selected.

Experiments are designed to assess the performance of the developed neighborhood search function under conditions that comply with and violate the stated assumptions based upon which the function is developed. The designed experiments are based on selected sets of standard benchmark instances that are commonly used in the literature of the flexible job shop scheduling problem. Those sets of problems add the flexibility ratio as an additional factor that is considered in the current experimental design. In addition, five different levels of the sizes of the sets that define the insertion positions and best found moves are defined in the experimental design.

Based on the conducted experiments, it is shown that the neighborhood search function provides a slightly better performance in the cases of low flexibility ratio. At low average setup time to processing time ratios, the function is found to achieve the same level of improvement capability compared to the case of higher ratios but with less number of iterations on average. The performance of the function is slightly improved when the triangular inequality is not satisfied in terms of the average percentage improvement. The best values for the number of local moves and insertion positions to randomly choose from are found to be 10 and 1 or 2 respectively. Another combination of 5 and 2 of those values is also a sufficiently good selection.

Based on the developed neighborhood search function, a tabu search metaheuristic is developed with minor modifications to the function that allow for excluding tabu moves from being reselected. The performance of the tabu search is assessed by comparing its makespan results with a lower bound. Results show that the tabu search approach can result in solutions with an average optimality gap below 10% at instances with low average setup time to processing time ratios. It is also shown that both the flexibility ratio and the triangular inequality conditions do not have significant effects on the quality of obtained solutions, for which the optimality gap is on average within 25.5%.

Future work that can build upon the results found in this paper and may include the development of a hybrid metaheuristic approach that combines the memory-based search of the developed tabu search approach with another population based metaheuristic such as particle swarm optimization and genetic algorithm. In addition, further research is needed to enhance the lower bounds.

References

[1] R. Graham, E. Lawler, J. Lenstra, A. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics 5 (1979) 287–326.

[2] P. Brandimarte, Routing and scheduling in a flexible job shop by tabu search, Annals of Operations Research 41 (1993) 157–183.

[3] K. Shanker, Y.-J. Tzen, A loading and dispatching problem in a random flexible manufacturing system, International Journal of Production Research 23 (3) (1985) 579–595.
[4] N. Nasr, E. A. Elsayed, Job shop scheduling with alternative machines, International Journal of Production Research 28 (9) (1990) 1595–1609.
[5] Y.-D. Kim, A comparison of dispatching rules for job shops with multiple identical jobs and alternative routeings, International Journal of Production Research 28 (5) (1990) 953–962.
[6] J. Hutchison, L. K., D. Snyder, P. Ward, Scheduling approaches for random job shop flexible manufacturing systems, International Journal of Production Research 29 (5) (1991) 1053–1067.
[7] E. Hurink, B. Jurisch, M. Thole, Tabu search for the job shop scheduling problem with multi-purpose machines, Operations Research-Spektrum 15 (1994) 205–215.
[8] S. Dauzère-pérès, J. Paulli, An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search, Annals of Operations Research 70 (1997) 281–306.
[9] M. Mastrolilli, L. Gambardella, Effective neighborhood functions for the flexible job shop problem, Journal of Scheduling 3 (1) (2000) 3–20.
[10] K. Mesghouni, S. Hammadi, P. Borne, Evolution programs for job shop scheduling, in: IEEE International Conference on Systems, Man, Cybernetics, Vol. 1, 1997, pp. 720–724.
[11] H. Chen, J. Ihlow, C. Lehmann, A genetic algorithm for flexible job-shop scheduling, in: Proceedings of the 1999 IEEE International Conference on robotics & Automation, 1999, pp. 1120–1125.
[12] J.-B. Yang, Ga-based discrete dynamic programming approach for scheduling in fms environments, IEEE Transactions on Systems, Man, and Cybernetics - Part B 31 (5) (2001) 824–835.
[13] I. Kacem, S. Hammadi, P. Borne, Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems, IEEE Transactions on Systems, Man, and Cybernetics - Part C 32 (1) (2002) 1–13.
[14] N. Ho, J. Tay, GENACE: an efficient cultural algorithm for solving the flexible job-shop problem, in: IEEE International conference on Robotics and Automation, 2004, pp. 1759–1766.
[15] S. L. Gao, J., M. Gen, A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems, Computers & Operations Research 35 (2008) 2892–2907.
[16] F. Pezzella, G. Morganti, G. Ciaschetti, A genetic algorithm for the flexible job-shop scheduling problem, Computers & Operations Research 35 (2008) 3202–3212.
[17] X. Wang, L. Gao, C. Zhang, X. Shao, A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem, The International Journal of Advanced Manufacturing Technology 51 (2010) 757–767. doi:10.1007/s00170-010-2642-2.
[18] G. Zhang, L. Gao, Y. Shi, An effective genetic algorithm for the flexible job-shop scheduling problem, Expert Systems with Applications 38 (2011) 3563–3573. doi:10.1016/j.eswa.2010.08.145.
[19] M. Yazdani, A. Amiri, M. Zandieh, Flexible job-shop scheduling with parallel variable neighborhood search algorithm, Expert Systems with Applications 37 (2010) 678–687.
[20] A. B. Hmida, M. Haouari, M.-J. Huguet, P. Lopez, Discrepancy search for the flexible job shop scheduling problem, Computers & Operations Research 37 (2010) 2192–2201.
[21] W. Bożejko, M. Uchroński, M. Wodecki, Parallel hybrid metaheuristics for the flexible job shop problem, Computers & Industrial Engineering 59 (2010) 323–333.
[22] Y. Yuan, H. Xu, Flexible job shop scheduling using hybrid differential evolution algorithms, Computers & Industrial Engineering 65 (2) (2013) 246–260. doi:10.1016/j.cie.2013.02.022.
[23] X. Zhu, W. Wilhelm, Scheduling and lot sizing with sequence-dependent setup: A literature review, IIE Transactions 38 (2006) 987–1007.
[24] A. Allahverdi, C. Ng, T. Cheng, M. Kovalyov, A survey of scheduling problems with setup times or costs, European Journal Operational Research 187 (2008) 985–1032.
[25] I.-C. Choi, D.-S. Choi, A local search algorithm for jobshop scheduling problems with alternative operations and sequence-dependent setups, Computers & Industrial Engineering 42 (2002) 43–58.
[26] K. Guimarães, M. Fernandes, An approach for flexible job-shop scheduling with separable sequence-dependent setup time, in: IEEE International Conference on Systems, Man, and Cybernetics, 2006, pp. 3727–3731.
[27] M. Saidi-Mehrabad, P. Fattahi, Flexible job shop scheduling with tabu search algorithms, International Journal of Advanced Manufacturing Technology 32 (2007) 563–570.
[28] F. M. Defersha, M. Chen, A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups, International Journal of Advanced Manufacturing Technology 49 (2010) 263–279.
[29] T. Abdelmaguid, Permutation-induced acyclic networks for the job shop scheduling problem, Applied Mathematical Modelling 33 (2009) 1560–1572.
[30] E. Balas, Machine sequencing via disjunctive graphs: an implicit enumeration algorithm, Operations Research 17 (1969) 941–957.
[31] B. Giffler, G. Thompson, Algorithms for solving production scheduling problems, Operations Research 8 (1960)

487–503.
[32] B. Jurisch, Lower bounds for the job-shop scheduling problem on multi-purpose machines, Discrete Applied Mathematics 58 (1995) 145–156.
[33] J. W. Barnes, J. B. Chambers, Flexible job shop scheduling by tabu search, Tech. Rep. ORP 96-09, Graduate Program in Operations and Industrial Engineering, The University of Texas at Austin (1996).
[34] I. Kacem, S. Hammadi, P. Borne, Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic, Mathematics and Computers in Simulation 60 (2002) 245–276.

Tables

Table 1: List of operations with routing and timing data

Operation index (i)	$\beta(i)$	$\{(m, p_i^m, s_{0,i}^m)$: $m \in \mu(i)\}$
1	J1	{(M1, 140, 9), (M2, 120, 16)}
2	J1	{(M1, 70, 9)}
3	J1	{(M3, 130, 10)}
4	J2	{(M2, 150, 10)}
5	J2	{(M1, 110, 5), (M3, 150, 20)}
6	J2	{(M3, 100, 20)}
7	J3	{(M3, 130, 19)}
8	J3	{(M1, 110, 24), (M2, 120, 20)}

Table 2: Sequence-dependent setup times between operations ($s_{i,j}^m$)

	M1								M2								M3							
i \j	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
1		2			7			4				12				8								
2					7			4																
3																					20	20	18	
4									12							5								
5	7	7						5											20			5	9	
6																			20				9	
7																			18		9	9		
8	4	4			5				8			5												

Table 3: References for standard FJSP benchmark instances

Reference	Acronym	Instances	Ranges for problem set parameters			
			$\|J\|$	$\|M\|$	n	*flex.*
Brandimarte [2] (problems mk01 to mk10)	BR	10	10-20	4-15	55-240	1.4-4.1
Barnes and Chambers [33]	BC	21	10-15	11-18	100-225	1.1-1.3
Dauzère-pérès and Paulli [8]	DP	18	10-20	5-10	196-387	1.1-5.0
Hurink et al. [7]	Hu-E	43	6-30	5-15	36-300	1.1-1.2
	Hu-R	43	6-30	5-15	36-300	1.88-2.06
	Hu-V	43	6-30	5-15	36-300	2.38-6.7
Kacem et al. [13, 34]	KA	5	4-15	5-10	12-56	5.0-10.0

Table 4: Selected problem instances for the low level of flexibility ratio

Referral #	Instance	Set acronym	Problem parameters			
			$\|J\|$	$\|M\|$	n	*flex.*
FJSP-L01	seti5x	BC	15	16	225	1.07
FJSP-L02	mt10x	BC	10	11	100	1.10
FJSP-L03	setb4x	BC	15	11	150	1.10
FJSP-L04	01a	DP	10	5	196	1.13
FJSP-L05	04a	DP	10	5	196	1.13
FJSP-L06	seti5xx	BC	15	17	225	1.13
FJSP-L07	la20	Hu-E	10	10	100	1.13
FJSP-L08	la13	Hu-E	20	5	100	1.14
FJSP-L09	la23	Hu-E	15	10	150	1.14
FJSP-L10	la22	Hu-E	15	10	150	1.15
FJSP-L11	la40	Hu-E	15	15	225	1.15
FJSP-L12	la25	Hu-E	15	10	150	1.16
FJSP-L13	la04	Hu-E	10	5	50	1.18
FJSP-L14	mt10xy	BC	10	12	100	1.20
FJSP-L15	setb4xy	BC	15	12	150	1.20
FJSP-L16	seti5xyz	BC	15	18	225	1.20
FJSP-L17	07a	DP	15	8	293	1.24
FJSP-L18	10a	DP	15	8	293	1.24
FJSP-L19	mt10xyz	BC	10	13	100	1.30
FJSP-L20	setb4xyz	BC	15	13	150	1.30
FJSP-L21	13a	DP	20	10	387	1.34
FJSP-L22	16a	DP	20	10	387	1.34
FJSP-L23	mk08	BR	20	10	225	1.43
FJSP-L24	05a	DP	10	5	196	1.69
FJSP-L25	la02	Hu-R	10	5	50	1.88
FJSP-L26	la06	Hu-R	15	5	75	1.88
FJSP-L27	la09	Hu-R	15	5	75	1.92
FJSP-L28	la17	Hu-R	10	10	100	1.93
FJSP-L29	mt20	Hu-R	20	5	100	1.94
FJSP-L30	la32	Hu-R	30	10	300	1.95

Table 5: Selected problem instances for the high level of flexibility ratio

Referral #	Instance	Set acronym	Problem parameters			
			$\|J\|$	$\|M\|$	n	*flex.*
FJSP-H01	la14	Hu-R	20	5	100	1.97
FJSP-H02	la21	Hu-R	15	10	150	2.01
FJSP-H03	la05	Hu-R	10	5	50	2.06
FJSP-H04	la04	Hu-V	10	5	50	2.38
FJSP-H05	11a	DP	15	8	293	2.42
FJSP-H06	la13	Hu-V	20	5	100	2.45
FJSP-H07	mk09	BR	20	10	240	2.53
FJSP-H08	03a	DP	10	5	196	2.56
FJSP-H09	la03	Hu-V	10	5	50	2.56
FJSP-H10	mt20	Hu-V	20	5	100	2.62
FJSP-H11	la10	Hu-V	15	5	75	2.67
FJSP-H12	la01	Hu-V	10	5	50	2.84
FJSP-H13	mt06	Hu-V	6	6	36	2.86
FJSP-H14	14a	DP	20	10	387	2.99
FJSP-H15	mk03	BR	15	8	150	3.01
FJSP-H16	mk06	BR	10	15	150	3.27
FJSP-H17	12a	DP	15	8	293	4.03
FJSP-H18	mt10	Hu-V	10	10	100	4.48
FJSP-H19	la32	Hu-V	30	10	300	4.54
FJSP-H20	la27	Hu-V	20	10	200	4.57
FJSP-H21	la34	Hu-V	30	10	300	4.62
FJSP-H22	la35	Hu-V	30	10	300	4.65
FJSP-H23	la16	Hu-V	10	10	100	4.70
FJSP-H24	la18	Hu-V	10	10	100	4.79
FJSP-H25	la24	Hu-V	15	10	150	4.85
FJSP-H26	15a	DP	20	10	387	5.02
FJSP-H27	18a	DP	20	10	387	5.02
FJSP-H28	la40	Hu-V	15	15	225	6.48
FJSP-H29	I_2	KA	10	7	29	7.00
FJSP-H30	I_3	KA	10	10	30	10.00

Table 6: Tabu search results for instances with low level of flexibility ratio

FJSP referral #	$LB^{(FJSP)}$	Satisfied triangular inequality				Unsatisfied triangular inequality			
		low s/p ratio		high s/p ratio		low s/p ratio		high s/p ratio	
		LB	RGTS	LB	RGTS	LB	RGTS	LB	RGTS
FJSP-L01	955	960	1234	1233	1779	960	1239	1168	1857
FJSP-L02	655	659	936	873	1360	658	939	813	1361
FJSP-L03	846	846	950	857	1440	846	960	860	1502
FJSP-L04	2505	2519	2625	3094	4273	2505	2665	2956	4207
FJSP-L05	2503	2534	2621	3189	4155	2512	2651	3088	4157
FJSP-L06	955	961	1225	1216	1756	960	1240	1189	1872
FJSP-L07	857	866	870	1103	1305	859	863	1115	1377
FJSP-L08	1053	1053	1072	1343	1673	1053	1069	1249	1639
FJSP-L09	950	950	977	1214	1500	950	986	1167	1532
FJSP-L10	832	832	903	1088	1376	832	906	976	1442
FJSP-L11	1034	1034	1181	1283	1718	1034	1178	1256	1795
FJSP-L12	894	894	960	1172	1458	894	958	1191	1502
FJSP-L13	568	568	581	792	929	568	574	749	922
FJSP-L14	655	659	911	868	1289	657	922	826	1363
FJSP-L15	845	845	936	880	1389	845	938	845	1450
FJSP-L16	955	960	1142	1204	1677	958	1154	1151	1763
FJSP-L17	2187	2200	2375	2657	3707	2187	2390	2568	3830
FJSP-L18	2178	2202	2427	2793	3817	2189	2424	2766	3849
FJSP-L19	655	659	920	868	1296	657	922	826	1401
FJSP-L20	838	838	927	891	1397	838	930	845	1423
FJSP-L21	2161	2171	2354	2688	3773	2161	2358	2587	4039
FJSP-L22	2148	2176	2371	2772	3853	2156	2403	2686	3861
FJSP-L23	523	529	538	807	911	523	526	808	870
FJSP-L24	2189	2211	2312	2764	3748	2189	2318	2692	3748
FJSP-L25	529	536	676	718	1036	537	679	760	1063
FJSP-L26	799	805	845	1107	1327	799	843	1033	1329
FJSP-L27	853	859	866	1033	1285	855	862	1023	1265
FJSP-L28	646	646	646	786	934	646	646	753	955
FJSP-L29	1022	1028	1114	1310	1798	1022	1106	1265	1745
FJSP-L30	1657	1657	1717	1984	2642	1657	1711	1890	2929

Table 7: Tabu search results for instances with high level of flexibility ratio

FJSP referral #	$LB^{(FJSP)}$	Satisfied triangular inequality				Unsatisfied triangular inequality			
		low s/p ratio		high s/p ratio		low s/p ratio		high s/p ratio	
		LB	RGTS	LB	RGTS	LB	RGTS	LB	RGTS
FJSP-H01	1070	1072	1079	1243	1550	1070	1076	1254	1505
FJSP-H02	808	809	878	892	1427	810	877	852	1424
FJSP-H03	457	458	468	584	793	458	469	561	769
FJSP-H04	502	502	508	513	758	502	505	508	755
FJSP-H05	2017	2026	2152	2332	3416	2026	2162	2303	3370
FJSP-H06	1038	1038	1048	1186	1499	1038	1044	1246	1511
FJSP-H07	299	299	318	299	500	299	312	299	503
FJSP-H08	2228	2236	2286	2486	3511	2232	2282	2415	3395
FJSP-H09	477	478	488	593	765	478	487	625	757
FJSP-H10	1022	1025	1027	1226	1437	1022	1025	1234	1390
FJSP-H11	804	806	811	965	1167	807	808	896	1119
FJSP-H12	570	573	586	659	924	572	585	710	892
FJSP-H13	47	47	47	53	57	47	47	53	55
FJSP-H14	2161	2168	2222	2361	3390	2165	2235	2387	3339
FJSP-H15	204	204	209	204	312	204	205	204	294
FJSP-H16	33	33	61	52	111	33	61	52	115
FJSP-H17	1969	1969	2106	1986	3213	1969	2118	1982	3177
FJSP-H18	655	655	655	719	852	655	655	690	856
FJSP-H19	1657	1657	1675	1657	2294	1657	1667	1657	2339
FJSP-H20	1084	1084	1100	1084	1572	1084	1095	1084	1576
FJSP-H21	1535	1535	1553	1535	2151	1535	1544	1535	2137
FJSP-H22	1549	1549	1566	1549	2168	1549	1557	1549	2162
FJSP-H23	717	717	718	818	872	717	717	795	911
FJSP-H24	663	663	663	740	865	663	663	717	858
FJSP-H25	773	773	789	773	1166	773	787	773	1183
FJSP-H26	2161	2161	2225	2161	3257	2161	2220	2161	3270
FJSP-H27	2057	2057	2195	2057	3299	2057	2212	2057	3210
FJSP-H28	955	955	955	1045	1229	955	955	1032	1247
FJSP-H29	15	15	15	15	34	15	15	15	31
FJSP-H30	7	7	9	10	19	7	7	10	19

Figure Captions

Figure 1. A sample solution: (a) Network representation, (b) Gantt chart

Figure 2. Interaction plot between n_{lm} and n_{Pos} for $\overline{imp\%}$

Figure 3. Interaction plot between n_{lm} and n_{Pos} for $\overline{itr^*}$

Figures

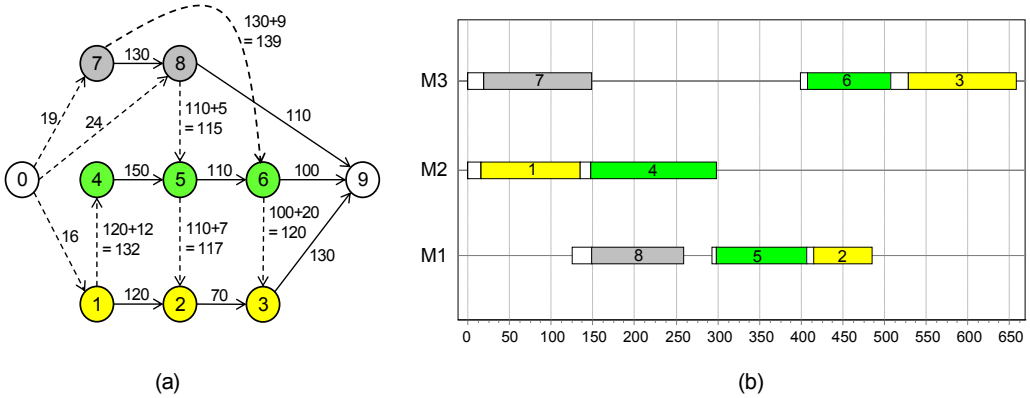

Figure 1: A sample solution: (a) Network representation, (b) Gantt chart

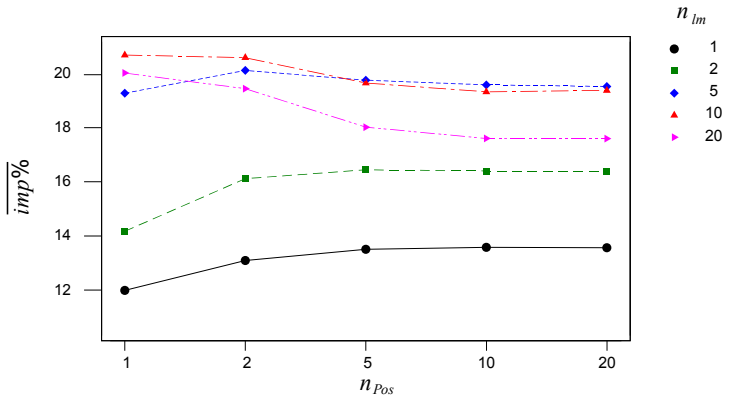

Figure 2: Interaction plot between n_{lm} and n_{Pos} for $\overline{imp\%}$

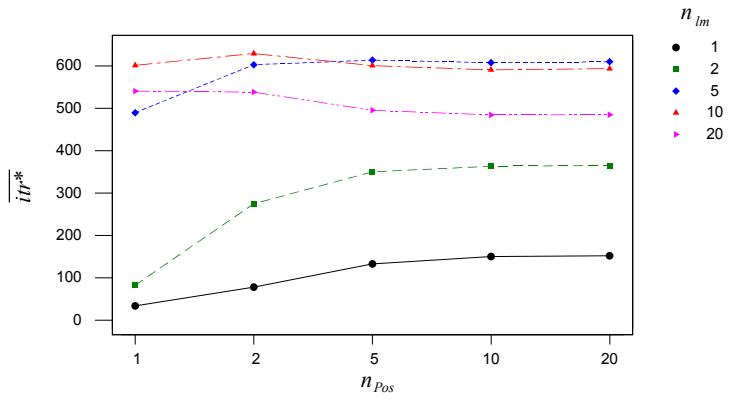

Figure 3: Interaction plot between n_{lm} and n_{Pos} for $\overline{itr^*}$