A variable neighbourhood search algorithm for the flexible job-shop scheduling problem

M. Amiri\(^a\); M. Zandieh\(^b\); M. Yazdani\(^c\); A. Bagheri\(^d\)

\(^a\) Department of Industrial Management, Management and Accounting Faculty, Allameh Tabatabaei University, Tehran, Iran \(^b\) Department of Industrial Management, Management and Accounting Faculty, Shahid Beheshti University, Tehran, Iran \(^c\) Faculty of Industrial and Mechanical Engineering, Qazvin Islamic Azad University, Qazvin, Iran \(^d\) Department of Industrial Engineering, Mazandaran University of Science and Technology, Babol, Iran

First published on: 26 August 2009


To link to this Article: DOI: 10.1080/00207540903055743

URL: http://dx.doi.org/10.1080/00207540903055743
A variable neighbourhood search algorithm for the flexible job-shop scheduling problem

M. Amiri\textsuperscript{a}, M. Zandieh\textsuperscript{b*}, M. Yazdani\textsuperscript{c} and A. Bagheri\textsuperscript{d}

\textsuperscript{a}Department of Industrial Management, Management and Accounting Faculty, Allameh Tabatabaei University, Tehran, Iran; \textsuperscript{b}Department of Industrial Management, Management and Accounting Faculty, Shahid Beheshti University, G.C., Tehran, Iran; \textsuperscript{c}Faculty of Industrial and Mechanical Engineering, Qazvin Islamic Azad University, Qazvin, Iran; \textsuperscript{d}Department of Industrial Engineering, Mazandaran University of Science and Technology, Babol, Iran

(Received 19 May 2008; final version received 18 May 2009)

The flexible job-shop scheduling problem (FJSP) is a generalisation of the classical job-shop scheduling problem which allows an operation of each job to be executed by any machine out of a set of available machines. FJSP consists of two sub-problems which are assigning each operation to a machine out of a set of capable machines (routing sub-problem) and sequencing the assigned operations on the machines (sequencing sub-problem). This paper proposes a variable neighbourhood search (VNS) algorithm that solves the FJSP to minimise makespan. In the process of the presented algorithm, various neighbourhood structures related to assignment and sequencing problems are used for generating neighbouring solutions. To compare our algorithm with previous ones, an extensive computational study on 181 benchmark problems has been conducted. The results obtained from the presented algorithm are quite comparable to those obtained by the best-known algorithms for FJSP.

Keywords: flexible job-shop scheduling; variable neighbourhood search; makespan; combinatorial optimisation

1. Introduction

There are a lot of scheduling problems that are fundamentally very hard, i.e., NP-hard. These problems cannot be formulated as linear programs and there are no simple rules or algorithms that yield optimal solutions in a limited amount of computer time (Pinedo 2005). One of the most difficult problems in this area is the job-shop scheduling problem (JSP). In JSP, a set of $n$ jobs must be processed on $m$ machines, where the processing of each job $i$ consists of $n_i$ operations performed on these machines. Each machine is continuously available from time zero and can process only one operation at a given time. The operations are processed on the machines without interruption. By considering the precedence constraints, this problem aims at finding the appropriate sequencing of the operations on the machines to optimise the performance indicator. Makespan is the time needed to complete all jobs, and can be considered as one of the performance indicators for JSP. It has also been demonstrated that JSP is NP-hard (Garey \textit{et al.} 1976).

*Corresponding author. Email: m_zandieh@sbu.ac.ir
The flexible job-shop scheduling problem (FJSP) is an extension of the classical JSP, which provides a closer approximation to a wide range of scheduling problems encountered in real manufacturing systems. In this problem, operations are allowed to be processed on any among a set of available machines. FJSP is more difficult than the classical JSP because it introduces a further decision level known as job routes besides the sequencing of operations on the machines. Determining the job routes means that we should decide which machine, among the available ones, must process each operation.

The high level of the complexity of FJSP demonstrates a cogent reason for using heuristic or meta-heuristic algorithms to optimise FJSP. Variable neighbourhood search (VNS) (Mladenovic and Hansen 1997, Hansen and Mladenovic 2001) is one of the renowned meta-heuristics which have been successfully applied to solve optimisation problems. It is capable of escaping from the local optima by systematic changes of the neighbourhood structures during the search process. In this paper, we develop a VNS algorithm for solving FJSP to minimise the makespan. The proposed VNS uses a linear encoding known as task sequencing list for representing solutions. In this method, two neighbourhood structures related to the sequencing problem and three neighbourhood structures related to the assignment problem are employed to generate neighbouring solutions. Another effectively useful operator in the searching solution space, which is applied in the present study, is a combined neighbourhood structure that generates a neighbouring solution by implementing changes in both assigning and sequencing of the candidate solution. Considering the aforementioned neighbourhood structures, the proposed optimisation method executes search process in the solution space. Furthermore, in this paper we tried to prove the efficiency and effectiveness of our algorithm by comparing the performance of the presented VNS with some previous algorithms.

The rest of the paper is organised as follows. Section 2 provides the literature review on FJSP. In Section 3, we present the formulation and constraints of the problem. Section 4 provides the encoding scheme and solution representation method. In Section 5, neighbourhood structures employed in the presented algorithm are explained in detail. Section 6 explains the basic structure of the VNS algorithm. Also, the proposed algorithm for solving FJSP is described in this section. In Section 7, the computational study performed with the proposed algorithm and its results are reported. The conclusion and some further research suggestions are presented in Section 8.

2. Literature review
The first research addressing FJSP was presented by Brucker and Schlie (1990). They developed a polynomial algorithm for solving this problem with two jobs. For solving the realistic case with more than two jobs, two types of approaches have been used: hierarchical approach and integrated approach. The hierarchical approach, in which assignment and sequencing are considered independently, is based on the idea of decomposing the original problem in order to reduce its complexity, whereas in the integrated approach the assignment and sequencing are investigated at the same time; in other words, assignment and sequencing are not separated. Furthermore, the integrated approach is much more difficult to solve, but, in general, achieves better results.

Regarding the implementation of these two approaches in the optimisation process, a variety of heuristic methods such as dispatching rules and local search and meta-heuristic
methods such as tabu search (TS), simulated annealing (SA), and genetic algorithm (GA) have been applied for solving FJSP in recent years.

Brandimarte (1993) applied a hierarchical approach for FJSP based on decomposition. He solved the routing sub-problem using some existing dispatching rules and then concentrated on the sequencing sub-problem which is solved by using a TS algorithm. Hurink et al. (1994) based on the integrated approach and Barnes and Chambers (1996) based on the hierarchical approach developed TS algorithms to solve FJSP. Dauzère-Pérès and Paulli (1997) defined a new neighbourhood structure for the problem where there was no distinction between reassigning and re-sequencing an operation, and proposed a TS algorithm based on the integrated approach. Mastrolilli and Gambardella (2000) improved Dauzère-Pérès TS techniques and presented two neighbourhood functions. Chen et al. (1999) developed a GA for FJSP. They split the chromosome representation into two parts, the first one defined the routing policy, and the second one delineated the sequence of operations on each machine. Kacem et al. (2002a, 2002b) proposed a localisation approach to solve the resource assignment problem, and an evolutionary approach controlled by the assignment model for FJSP. Zhang and Gen (2005) proposed a multistage operation-based GA to deal with the problem from the point of view of dynamic programming. Xia and Wu (2005) worked on a hierarchical approach for solving multi-objective FJSP. The approach made use of a particle swarm optimisation (PSO) to assign operations to machines and an SA algorithm to schedule operations on each machine.

Chan et al. (2006) developed an innovative GA-based approach, namely iterative GA (IGA), for solving resource-constrained FJSP (RCFJSP). Fattahi et al. (2007) proposed a mathematical model and two meta-heuristic algorithms (SA and TS) for solving FJSP. Moreover, considering two developed meta-heuristics and concerning the integrated and hierarchical approaches, they presented six different algorithms. Pezzella et al. (2008) propounded a GA for FJSP based on the integrated approach, in which a mix of different strategies for generating the initial population, selecting individuals for reproduction, and reproducing new individuals is presented. Shi-Jin et al. (2008) developed a filtered-beam-search-based heuristic algorithm (HFBS) to find suboptimal schedules within a reasonable computational time for the FJSP with multiple objectives of minimising the makespan, the total workload of machines, and the workload of the most loaded machine. Tay and Ho (2008) solved the multi-objective FJSP using dispatching rules discovered through genetic programming. They also applied an integrated approach to devise their algorithm. Experimental results showed that composite dispatching rules generated by the genetic programming framework outperforms the single dispatching rules and the composite dispatching rules selected from the literature with respect to minimum makespan, mean tardiness, and mean flow time objectives. Gao et al. (2008) studied the FJSP with three objectives: min makespan, min maximal machine workload and min total workload. They developed a hybrid genetic algorithm (hGA) based on the integrated approach for this problem. Under the framework of the GA, variable neighbourhood descent (VND) is applied to each newly generated offspring to improve its quality before injecting it into the population.

3. Problem definition

The FJSP is to organise the execution of \( n \) jobs on \( m \) machines. In this problem, there are a set of machines \( Ma = \{M_1, M_2, \ldots, M_m\} \), and a set of jobs \( J = \{J_1, J_2, \ldots, J_n\} \). Each job \( J_i \)
consists of a sequence of \( n_i \) operations \((O_{i1}, O_{i2}, \ldots, O_{in_i})\) to be performed one after the other according to the given sequence. Each operation \( O_{ij} \), i.e., the operation \( j \) of job \( i \), can be processed on any among a subset of compatible machines called \( M_{aij} \in Ma \). Flexibility of problems can be categorised into partial flexibility and total flexibility. If \( M_{aij} = Ma \) for all operations, we have a total flexibility, otherwise flexibility would be partial. If \( f \) represents the average number of available machines per operation in one problem and \( m \) stands for the number of machines for this problem, the proportion of these factors, \( R_f = f/m \), will be an appropriate criterion for comparing flexibility among FJSP problems.

The conditions and constraints in this problem are as follows:

1. Machines are independent from each other.
2. Jobs are independent from each other. There are no precedence constraints among the operations of different jobs. Nonetheless, there are precedence constraints among the operations of the same job.
3. All jobs and machines are available at time 0.
4. Setup time of machines and move time between operations are negligible.
5. At a given time, a machine can only execute one operation.
6. Pre-emption is not allowed. That is, each operation must be completed without interruption, once it starts.
7. The processing time of operation \( O_{ij} \) on machine \( Mr \) is \( P_{i,j,r} > 0 \).

The problem is to assign each operation to an appropriate machine out of a set of capable machines, and to sequence the operations on the machines in order to minimise the makespan, i.e., the time needed to complete all jobs, which is defined as \( C_{\text{max}} = \max(C_i) \), where \( C_i \) is the completion time of job \( J_i \).

4. Encoding scheme

The FJSP is a combination of machine assignment and operation sequencing decisions. Thus, a solution can be expressed by the assignment of the operations to the machines, and the processing sequence of the operations on the machines. In this paper, to represent our VNS algorithm solutions, we use the task sequencing list provided by Kacem et al. (2002a), in which a string embodies scheduling operations. Each operation in the solution string is represented by a triple \((i, j, a(i, j))\), where \( i \) represents the job which an operation belongs to, \( j \) characterises the progressive number of that operation within job \( i \), and \( a(i, j) \) indicates the machine assigned to that operation. The length of the string is equal to the total number of operations of jobs. The left-to-right ordering of operations in the solution string represents the sequencing of the operations on the machines. Additionally, the operations of the same job should be located according to their progressive number in the solution string.

Data related to a given problem with total flexibility is shown in Table 1, in which rows correspond to operations and columns represent machines. The solution \( S = (O_{21}, M_2), (O_{22}, M_4), (O_{41}, M_1), (O_{31}, M_1), (O_{11}, M_4), (O_{32}, M_3), (O_{42}, M_4), (O_{12}, M_3), (O_{43}, M_2), (O_{13}, M_1) \) is one of the feasible solutions from the solution space for the mentioned problem. The representation of this solution in the task sequencing list format is provided in Figure 1. Also, a Gantt chart of the solution is shown in Figure 2 where \( C_{\text{max}} = 15 \).
5. Neighbourhood structure

The main purpose of applying a neighbourhood structure is to produce a neighbouring solution from the current solution via making some changes in it. A variety of neighbourhood structures have been applied to scheduling problems. These neighbourhood structures must work so that they prevent any infeasible solution. In this section, six types of neighbourhood structures employed in the proposed algorithm are presented.

Table 1. Processing time table.

<table>
<thead>
<tr>
<th></th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_{11}$</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>$O_{12}$</td>
<td>8</td>
<td>6</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>$O_{13}$</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$O_{21}$</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>$O_{22}$</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>$O_{31}$</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>$O_{32}$</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$O_{41}$</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>$O_{42}$</td>
<td>9</td>
<td>8</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>$O_{43}$</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

$S : (2,1,2) (2,2,4) (4,1,1) (3,1,1) (1,1,4) (3,2,3) (4,2,4) (1,2,3) (4,3,2) (1,3,1)$

Figure 1. Representation of solution $S$.

Figure 2. Gantt chart.

5. Neighbourhood structure

The main purpose of applying a neighbourhood structure is to produce a neighbouring solution from the current solution via making some changes in it. A variety of neighbourhood structures have been applied to scheduling problems. These neighbourhood structures must work so that they prevent any infeasible solution. In this section, six types of neighbourhood structures employed in the proposed algorithm are presented.
5.1 Neighbourhood structure $Se_1$

To change the sequence of the operations in the candidate solution string, neighbourhood structure $Se_1$ is utilised so that the assignment of the operations to the machines remains unchanged. Regarding the precedence constraints among the operations of the same job, to generate a neighbouring solution, this neighbourhood structure makes some changes in the candidate solution string by replacing the positions of some operations with their neighbouring ones. The steps of this neighbourhood structure are spelled out below:

First, the iteration counter of this neighbourhood structure, $G$, begins with 1 ($G \leftarrow 1$). In that case, two steps $a$ and $b$ are then performed: ($a$) one cell among the existing cells in the candidate solution string is selected; and ($b$) if the move is feasible in respect of precedence constraints, the operation of the selected cell is substituted for the operation of its preceding cell. Otherwise, the operation of the selected cell is substituted for the operation of its succeeding cell. Furthermore, regarding precedence constraints, it is possible that in step $b$ the operations of the selected cell would not be replaced with their preceding and succeeding cells. After steps $a$ and $b$ are performed, the first iteration is accomplished and the next one begins ($G \leftarrow G + 1$). In the next iterations, steps $a$ and $b$ are performed like the first iteration. The process of this neighbourhood structure continues until the maximum number of iterations related to it, i.e., $G_{\text{max}}$, is met. Neighbourhood structure $Se_1$ with various measures of $G_{\text{max}}$ can be utilised in the presented algorithm.

Consider the following example in Figure 3. $S_1$ is a candidate solution for generating neighbouring solution and $G_{\text{max}}$ is set to be 2. The iteration counter $G$ begins with 1 and one cell is selected from the solution string at random. Suppose the second cell including operation $O_{2,1}$ is selected for replacement; then, the operation of the selected cell is substituted for the operation in cell one. Now, one unit is added to the iteration counter $G$, ($G \leftarrow 2$). Then, another cell is selected from the solution string at random. Supposedly, cell 8 containing operation $O_{4,2}$ is chosen. The operation of the selected cell will not be substituted for the operation of its preceding cell, but it can be substituted for the succeeding one. Therefore, the operation in cell 8 is substituted for the operation in cell 9. As a result, with respect to the execution of two iterations, process of this neighbourhood structure is accomplished and the neighbouring solution $S'_1$ is gained after two changes.

5.2 Neighbourhood structure $As$

To generate a neighbouring solution, neighbourhood structure $As$ changes the assignment of the operations to the machines in the candidate solution string. In this neighbourhood

\[
\begin{array}{cccccccc}
(3,1,1) & (2,1,4) & (1,1,2) & (2,2,3) & (1,2,1) & (1,3,3) & (4,1,4) & (4,2,2) & (3,2,1) & (4,3,2) \\
(2,1,4) & (3,1,1) & (1,1,2) & (2,2,3) & (1,2,1) & (1,3,3) & (4,1,4) & (4,2,2) & (3,2,1) & (4,3,2) \\
(2,1,4) & (3,1,1) & (1,1,2) & (2,2,3) & (1,2,1) & (1,3,3) & (4,1,4) & (4,2,2) & (3,2,1) & (4,3,2)
\end{array}
\]
structure, sequencing of the operations on the machines does not change. The steps of
neighbourhood structure $A_s$ are presented in the following.

First, the iteration counter of this neighbourhood structure, $A$, starts from 1 ($A \leftarrow 1$). In
case an operation is randomly chosen out of the operations of the candidate
solution string which are capable of being processed on more than one machine.
Subsequently, changing the assignment of the selected operation starts. Among the
available machines for the selected operation, a new machine is chosen at random and this
operation will be assigned to it. Accordingly, the first iteration will be accomplished, and
another iteration begins ($A \leftarrow A + 1$). The process of this neighbourhood structure
continues until the maximum number of iterations related to it, i.e., $A_{max}$ is met.
Neighbourhood structure $A_s$ is capable of using dissimilar measures of $A_{max}$ in the
presented algorithm.

5.3 Neighbourhood structure Co
Neighbourhood structure Co is a combination of neighbourhood structure $S_e_1$ and
neighbourhood structure $A_s$. To generate a neighbouring solution by this neighbourhood
structure, at first neighbourhood structure $S_e_1$ with adjusted $G_{max}$, and then neighbour-
hood structure $A_s$ with adjusted $A_{max}$ are executed on the candidate solution string. The
solution obtained by employing these two neighbourhood structures is regarded as the
neighbouring solution.

5.4 Neighbourhood structure $S_e_2$
Neighbourhood structure $S_e_2$ substitutes the positions of operations concerning two jobs
in the candidate solution string to produce a neighbouring solution, where the assignment
of operations on the machines is preserved. This process is described below.

First, two jobs are chosen at random from jobs of the candidate solution string. If the
number of operations is equal in two selected jobs, in order to generate a neighbouring
solution, the places of operations of two jobs in the candidate solution string are, in a one-
to-one correlation, swapped with regard to the progressive number. On the contrary, if the
number of operations is not equal in two selected jobs, in the new solution string, the
operations of the job with a fewer number of operations move to the cells of the job with
more number of operations in respect of sequencing constraints. Then, the operations
which are related to the job with a greater number of operations can occupy both their
remaining cells and the whole previous cells of the job with a fewer number of operations
with regard to sequencing constraints. Consequently, the neighbouring solution will be
achieved. To clarify the above-mentioned process, an example of the execution process of
this neighbourhood structure is provided in Figure 4. Note that $S_2$ is a candidate solution
for generating neighbouring solution and jobs 1 and 3 are selected for substitution.

\[ S_2 : (1,1,2) \begin{array}{|c|c|c|c|c|c|c|c|} \hline (1,1,1) & (2,1,4) & (4,1,4) & (2,2,3) & (1,2,1) & (4,2,2) & (1,3,3) & (3,2,1) & (4,3,2) \hline \end{array} \]

\[ S_2' : (3,1,1) \begin{array}{|c|c|c|c|c|c|c|c|} \hline (1,1,2) & (2,1,4) & (4,1,4) & (2,2,3) & (3,2,1) & (4,2,2) & (1,2,1) & (1,3,3) & (4,3,2) \hline \end{array} \]

Figure 4. Example for neighbourhood structure $S_e_2$. 

5.5 Intelligent neighbourhood structure $I_1$

Neighbourhood structure $I_1$ has been adapted to intelligent mutation presented by Pezzella et al. (2008). The following describes the steps of this neighbourhood structure.

At first, machines with the most and least workloads from the candidate solution string are selected. Workload of the machine is the sum of the processing times of the operations assigned to that machine. Afterwards, for creating a neighbouring solution, an operation is randomly chosen from the machine having the most workload and is assigned to the machine with the least workload. In this neighbourhood structure, if the machine with the least workload is not suitable for processing the selected operation, another machine will be selected at random out of a set of available machines for this operation. The previous assignment does not change provided that the selected operation can only be processed on one machine.

5.6 Intelligent neighbourhood structure $I_2$

To generate a neighbouring solution with neighbourhood structure $I_2$, an operation is randomly chosen from the machine which spends the maximum time (equal to makespan) to complete its assigned operations, and this operation is then assigned to the machine spending the minimum time to complete its assigned operations. In this neighbourhood structure, if the selected operation cannot be processed on the new machine, another machine will be chosen at random out of a set of capable machines for this operation. If the selected operation is only capable of being processed on one machine, the prior assignment to it will be kept fixed.

6. A variable neighbourhood search for FJSP

Variable neighbourhood search was first proposed by Mladenovic (1995). VNS is a recent meta-heuristic which exploits systematically the idea of neighbourhood change, both in descent to local minima and in escape from the valleys which contain them (Hansen et al. 2006). This meta-heuristic has been successfully applied to diverse combinatorial optimisation problems such as P-median problem (Hansen and Mladenovic 1997), multi-source problem (Brimberg et al. 2000), minimum spanning tree problem (Ribeiro and Souza 2002), clustering problem (Hansen and Mladenovic 2002), graph colouring problem (Avanthay et al. 2003), redundancy allocation problem (Liang and Chen 2007), and job-shop scheduling problem (Sevkli and Aydin 2007). The basic steps of VNS meta-heuristic are shown in Figure 5.

During the initialisation step, a set of neighbourhood structures ($N_k$, for $k = 1, \ldots, k_{\text{max}}$) and the sequence of their implementations are determined. The performance of this meta-heuristic significantly depends on the efficiency of the neighbourhood structures. In this step, the stopping condition is delineated. In addition, an initial solution is generated and is set as the current solution $x$. In the search loop (step 3), the shake procedure is implemented for randomly generating a neighbouring solution $x'$ from the current solution $x$ based on the first neighbourhood structure $N_1$. Then a local search is carried out for obtaining local optimum $x''$, with $x'$ as input solution of the local search. Afterwards, local optimum $x''$ is compared with current solution $x$ in terms of the solution quality. If $x''$ is better than $x$, i.e., improved solution is obtained, $x$ is replaced with $x''$ and the search begins again at the first neighbourhood with the updated $x$. Otherwise, the search loop is
iterated by the next neighbourhood \( (k \leftarrow k + 1) \). In this case, the neighbourhood structure is systematically changed and the shake procedure works to switch to another region of the search space so as to carry out a new local search there. After all neighbourhoods are considered and no further improvement can be obtained for the current solution \( x \), the next iteration of the algorithm is started and the search begins again at the first neighbourhood of current solution \( x \). The VNS algorithm continues until a stopping condition is satisfied. This stopping condition will usually be the maximum computational time since the last improvement, or the maximal allowed CPU time, or the maximum number of iterations. The rest of this section is subdivided into the following subdivisions.

We describe, in Section 6.1, how the initial solution is generated; and in Section 6.2 a comprehensive description of the proposed VNS algorithm is given.

### 6.1 Initial solution

At first, an initial population should be generated. For generating initial assignments, two approaches presented by Pezzella \textit{et al.} (2008) are used:

**Assignment rule 1:** search for the global minimum in the processing time table.

**Assignment rule 2:** randomly permute jobs and machines in the processing time table.

Both approaches are modified versions of the localisation approach by Kacem \textit{et al.} (2002a). The implementation of a mix of these two rules generates the initial set of assignments. Thirty per cent of initial assignments can be generated by rule 1 and 70% by rule 2. Once the initial assignments are generated, we have to determine how to sequence the operations on the machines. Obviously, the sequencing is feasible if it respects the precedence constraints among the operations of the same job. The sequencing of the initial assignments is obtained by random selection of a job, being known as a dispatching rule.
Consequently, the initial population is achieved due to applying the aforementioned procedure in the given order. It should be mentioned that the number of the generated initial population is set to be 1000. Finally, the best solution is chosen out of the initial population for initial solution of the algorithm.

In the experiments related to optimising problems, at first we tested the random methods for generating initial solution of the presented algorithm and then in the subsequent test, we employed the above-mentioned method in the article to generate an initial solution. The comparisons between initial and final solutions in these two tests revealed that the presented algorithm almost attained better results when the discussed method in the article was applied for generating initial solution.

### 6.2 Proposed algorithm

Figure 6 demonstrates the general process of the proposed VNS algorithm. In the initialisation step, the neighbourhood structures needed for searching the solution space are identified. Two sets of neighbourhood structures are used in the presented algorithm: $N_k^s$ and $N_l^l$. Neighbourhood structures related to $N_k^s$ are employed in the shake procedure and neighbourhood structures related to $N_l^l$ are utilised in the local search procedure. The stop condition of the algorithm equalling to maximum number of iterations is also identified in the initialisation step. The initial solution $x$ is generated in this step by the strategy presented in Section 6.1. Besides, $\tilde{x} \leftarrow x$, in which $\tilde{x}$ is the representative of the current solution in the various steps of the algorithm. To measure the quality of solutions, objective function value is considered as makespan.

The main part of the proposed algorithm is comprised of internal and external loops. The internal loop is responsible for searching the solution space, whereas the external loop controls the stop condition of the algorithm. The symbol $k$ is delineated as the iteration counter of the internal loop, and the maximum number of iterations of the internal loop, i.e., $k_{\max}$ is set to be 3.

In each of the three iterations of the internal loop, at first, a shake procedure is implemented for randomly generating a neighbouring solution $x'$ from the current solution $\tilde{x}$. Table 2 indicates the neighbourhood structure employed in the shake procedure for each iteration of the internal loop. In all iterations, neighbourhood structure $Co$ has been used and set measures of $G_{\max}$ and $A_{\max}$ for the employed structure increase with regard to the change in the iteration counter $k$ from 1 to 3. As a result, the shake procedures in these three iterations are different in terms of the number of changes in the solution.

In the next step of each iteration of the internal loop, a local search procedure should be performed considering the solution obtained from the shake procedure. The local search procedure is shown in Figure 6. Neighbourhood structure $Se_1$ with four different values of $G_{\max}$, neighbourhood structure $As$ with 2 different values of $A_{\max}$, and neighbourhood structures $Se_2$, $I_1$, and $I_2$ are used in this step. These neighbourhood structures are applied according to counter $l$, and $l_{\max}$ is set to be 9. The implementing order of the neighbourhood structures in the local search procedure is determined based on the type and number of necessary changes for appropriate and diverse searches in the solution space and on results related to testing different sequences of these structures. By the commencement of the local search procedure, $l$ and $n$ (iteration counter of local search) begin with 1. Then, the first neighbourhood structure related to the local search procedure is employed to generate neighbouring solution $xp$ from input solution $x'$. In this case $xp$ is
Initialisation:
a. Select the set of neighbourhood structures $N^k_i$, for $k = 1, \ldots, k_{max}$, that will be used in the shaking step, and the set of neighbourhood structures $N^{l}_i$, for $l = 1, \ldots, l_{max}$ that will be used in the local search step.
b. Generate a population of solutions and the best solution among them $x$ is selected for initial solution.
c. Choose stopping condition (maximum number of iterations).
d. Set $\bar{x} \leftarrow x$.

Repeat (external loop)
Set $k \leftarrow 1$;
Repeat (internal loop)
Shaking:
Generate random point $x' \in N_i^k(\bar{x}) \leftarrow Co(\bar{x})$;
Local search:
Get solution, $x'$;
Set $n \leftarrow n+1$ and $l \leftarrow 1$;
While $n \leq n_{max}$ do
if $l = 1$ then $xp \in N^{l}_i(x') \leftarrow S_1(x')$; (i.e., $G_{max} = 3$)  
else if $l = 2$ then $xp \in N^{l}_i(x') \leftarrow A_1(x')$; (i.e., $A_{max} = 1$)  
else if $l = 3$ then $xp \in N^{l}_i(x') \leftarrow S_2(x')$; (i.e., $G_{max} = 5$)  
else if $l = 4$ then $xp \in N^{l}_i(x') \leftarrow A_2(x')$; (i.e., $A_{max} = 2$)  
else if $l = 5$ then $xp \in N^{l}_i(x') \leftarrow S_3(x')$; (i.e., $G_{max} = 7$)  
else if $l = 6$ then $xp \in N^{l}_i(x') \leftarrow I_1(x')$;  
else if $l = 7$ then $xp \in N^{l}_i(x') \leftarrow S_2(x')$; (i.e., $G_{max} = 9$)  
else if $l = 8$ then $xp \in N^{l}_i(x') \leftarrow I_2(x')$;  
else if $l = 9$ then $xp \in N^{l}_i(x') \leftarrow S_3(x')$;
endif
if $f(xp) \leq f(x')$ then $x' \leftarrow xp$ and $l \leftarrow l+1$; else $l \leftarrow l+1$;
endif
if $l = 10$ then $l \leftarrow 1$;
endif
$n \leftarrow n+1$;
endWhile
$x^* \leftarrow x'$;

Updating:
if $f(x^*)$ is better than $f(\bar{x})$ then $\bar{x} \leftarrow x^*$, and $k \leftarrow 1$; else $k \leftarrow k+1$;
endif
Until $k > k_{max}$ ($k_{max} = 3$)
Until the stopping condition is met

Figure 6. Proposed VNS algorithm with pseudo-code.

Table 2. Neighbourhood structure employed in the shake procedure for each iteration of the internal loop.

<table>
<thead>
<tr>
<th>Step</th>
<th>$k = 1$</th>
<th>$k = 2$</th>
<th>$k = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neighbourhood structure</td>
<td>$Co$</td>
<td>$Co$</td>
<td>$Co$</td>
</tr>
<tr>
<td>$G_{max}$</td>
<td>3</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>$A_{max}$</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
compared with \( x' \). If \( xp \) is better than \( x' \) or both of them are equal in terms of the solution quality (makespan), \( x' \) is replaced with \( xp \) (\( x' \leftarrow xp \)), whereas the present neighbourhood structure is preserved (\( l \leftarrow l \)). Also, when \( xp \) is worse than \( x' \), \( x' \) is preferred and one unit is added to the counter \( l (l \leftarrow l + 1) \) and the next neighbourhood structure is activated. When the first iteration of the local search is accomplished, one unit is added to counter \( n (n \leftarrow n + 1) \) and the next iteration begins. During the local search procedure, if counter \( l \) equals to \( l_{\text{max}} + 1 \), the first neighbourhood structure is used again (\( l \leftarrow 1 \)). The number of neighbourhood searches in the local search procedure is set to be \( n_{\text{max}} \) and this procedure continues until all iterations of it are completed. The obtained solution from the local search procedure is placed in \( x'' \).

Then the updating step is commenced. In this step, \( x'' \) is compared with \( \bar{x} \) in terms of the solution quality. If \( x'' \) is better than \( \bar{x} \), i.e., another local optimum is acquired, \( \bar{x} \) is replaced with \( x'' \) (\( \bar{x} \leftarrow x'' \)) and the search process begins again at the first iteration of the internal loop (\( k \leftarrow 1 \)). Otherwise, one unit is added to the iteration counter \( k (k \leftarrow k + 1) \) and the next iteration of the internal loop starts. On condition that all three iterations of the internal loop are considered, one iteration of the VNS algorithm is accomplished. In that case, the stop condition is checked by the external loop. If the stop condition is not met, the next iteration of the algorithm begins. The VNS algorithm continues until the prespecified maximum number of iterations of the algorithm is achieved. When the process of the algorithm stops, the final current solution is used as the best solution \( x^* \).

Regarding the test on different values for algorithm parameters and considering the computational results, the following settings are adjusted for the presented VNS algorithm:

- The maximum number of iterations of the algorithm (stop condition) is limited to 5000.
- Number of neighbourhood searches in the local search procedure \( (n_{\text{max}}) \) is set to be 500.

### 7. Computational results

This section describes the computational tests which were used to evaluate the effectiveness and efficiency of the proposed VNS algorithm. In order to conduct the experiment, we implemented the algorithm in C++ language and run on a PC with 2.0 GHz and 1.0 GB of RAM memory. The performance of the proposed algorithm was compared with that of other algorithms. The non-deterministic nature of the presented algorithm made it necessary to carry out multiple runs on the same problem instance in order to obtain meaningful results. Therefore, the best solution was selected for each problem after five runs of the VNS algorithm from different initial solutions. The set of problems under investigation is described below:

1. Kacem data: the data set is a set of three problems (problem 8\( \times 8 \), problem 10\( \times 10 \) and problem 15\( \times 10 \)) which are all taken from Kacem et al. (2002a, 2002b). Problem 8\( \times 8 \) is an instance of partial flexibility that consists of eight jobs with 27 operations which can be processed on eight machines, problem 10\( \times 10 \) has total flexibility that is composed of 10 jobs with 30 operations which can be implemented on 10 machines, and problem 15\( \times 10 \) has total flexibility that is
comprised of 15 jobs with 56 operations which can be performed on 10 machines. The details about the Kacem data can be found in Xia and Wu (2005).

(2) BRdata: the data set consists of 10 test problems from Brandimarte (1993). The data was randomly generated using a uniform distribution between given limits. The number of jobs ranges from 10 to 20, the number of machines ranges from 4 to 15, the number of operations for each job ranges from 5 to 15, and the number of operations for all jobs ranges from 55 to 240.

(3) BCdata: the data set is a set of 21 problems from Barnes and Chambers (1996). The data was constructed from three of the most challenging classical job-shop scheduling problems (mt10, la24, la40) (Fisher and Thompson 1963, Lawrence 1984) by replicating machines selected according to two simple criteria: the total processing time required by a machine and the cardinality of the critical operations on a machine. The number of jobs ranges from 10 to 15, the number of machines ranges from 11 to 18, the number of operations for each job ranges from 10 to 15, and the number of operations for all jobs ranges from 100 to 225.

(4) DPdata: the data set consists of 18 problems from Dauzére-Péres and Paulli (1997). The number of jobs ranges from 10 to 20, the number of machines ranges from 5 to 10, the number of operations for each job ranges from 5 to 25, and the number of operations for all jobs ranges from 100 to 225. A set of machines being capable of performing an operation was constructed by letting a machine be in that set with a probability that ranges from 0.1 to 0.5.

(5) HUdata: HUdata is a set of 129 problems from Hurink et al. (1994). The problems were obtained from three problems (mt06, mt10, mt20) by Fisher and Thompson (1963) and 40 problems (la01-la40) from Lawrence (1984). Depending on the average number of alternative machines for each operation, HUdata was divided into three subsets: Edata, Rdata and Vdata. The number of jobs ranges from 6 to 30, the number of machines ranges from 5 to 15, the number of operations for each job ranges from 5 to 15, and the number of operations for all jobs ranges from 36 to 300.

The first data set under investigation belongs to the Kacem data. Table 3 compares the result of the presented VNS algorithm with the following algorithms: AL+CGA of Kacem et al. (2002a, 2002b), PSO+SA of Xia and Wu (2005), moGA of Zhang and Gen (2005), and hybrid genetic algorithm (hGA) of Gao et al. (2008). The first column characterises the size of the problem, in which \( n \) stands for the number of jobs and \( m \) embodies the number of given machines in the problem. The second column up to the fifth one represent the best makespans resulted from AL+CGA, PSO+SA, moGA, and hGA respectively.

<table>
<thead>
<tr>
<th>Problem ( n \times m )</th>
<th>AL + CGA</th>
<th>PSO + SA</th>
<th>moGA</th>
<th>hGA</th>
<th>( BC_{\text{max}} )</th>
<th>AV(( C_{\text{max}} ))</th>
<th>AV(CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 \times 8</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>1.46</td>
</tr>
<tr>
<td>10 \times 10</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>1.73</td>
</tr>
<tr>
<td>15 \times 10</td>
<td>24</td>
<td>12</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>4.59</td>
</tr>
</tbody>
</table>
The sixth column signifies the best makespan obtained from five runs of the VNS algorithm, and the seventh column reports the average of the obtained results of five runs. The eighth column represents the average of CPU times for five runs of the presented algorithm in terms of seconds. The results show that we have achieved the best solution that has ever been obtained for problems $8 \times 8$ and $10 \times 10$. In the case of the problem $15 \times 10$, there is just one time unit difference between the best makespan obtained from our algorithm and the one presented by hGA of Gao et al. (2008).

Another data set which was studied included BRdata problems. In Table 4, we compared the results of our VNS to the results of the GA of Pezzella et al. (2008), TS of Mastrolilli and Gambardella (2000), and hGA of Gao et al. (2008) on 10 FJSP instances from Brandimarte. The first and second columns symbolise the name and size of the problem respectively. In the third column, the average number of available machines per operation ($f$) is shown for each problem. In the fourth column, the criterion of flexibility $R_f$, which was described in the problem definition section, is presented for each problem.

In the fifth column, (LB, UB) denotes the optimum makespan if known, otherwise, it is the best lower and upper bound that has ever been found. The sixth column refers to the best makespan resulted from five runs of the VNS algorithm, and the average of the obtained results of five runs are shown in the seventh column. The eighth column represents average of CPU times for five runs of the presented algorithm in terms of seconds. Columns nine, 11, and 13 represent the best results obtained by GA, TS, and hGA respectively. Relative deviation criterion is used to compare the results of the presented algorithm with those of the three above-mentioned ones. Relative deviation is obtained as follows:

$$dev = \left(\frac{BC_{\text{max}}(\text{comp}) - BC_{\text{max}}(VNS)}{BC_{\text{max}}(\text{comp})}\right) \times 100\%,$$

where $BC_{\text{max}}(VNS)$ is the best makespan obtained by our algorithm and $BC_{\text{max}}(\text{comp})$ is the best makespan of the algorithm that we compare ours to. Measures concerning relative deviation are presented in columns 10, 12, and 14. The results of Table 4 revealed that our algorithm worked better than GA of Pezzella et al. (2008) in BRdata problems. The results of our algorithm were also comparable to the best results of TS of Mastrolilli and Gambardella (2000). But the proposed algorithm is rather less efficient than hGA of Gao et al. (2008) in BRdata problems.

In Table 5, we compared the average of results of the presented VNS to the average of results of TS of Mastrolilli and Gambardella (2000), and hGA of Gao et al. (2008) on 10 FJSP instances from Brandimarte.

It should be noted that we did not consider making comparisons among algorithms based on CPU time criterion due to lack of information on CPU times for some of the algorithms discussed in this section (AL+CGA, PSO+SA, moGA and GA), and on the features of computers used in executing some of these algorithms.

Table 6 encompasses the computational results of four groups of problems with regard to the mean relative error (MRE) criterion. The first column reports the data set, the second column reports the number of instances for each class, and the next five columns report the mean relative error of the best solution obtained by our VNS, by GA of Pezzella et al. (2008), by GA of Chen et al. (1999), by TS of Mastrolilli and Gambardella (2000), and by hGA of Gao et al. (2008), respectively, with respect to the best-known lower bound. The relative error ($RE$) is calculated as follows:

$$RE = \left(\frac{BC_{\text{max}} - LB}{LB}\right) \times 100\%,$$
Table 4. Comparison of the presented VNS algorithm with the three algorithms (GA, TS, hGA) on BRdata.

<table>
<thead>
<tr>
<th>Problem</th>
<th>$n \times m$</th>
<th>$f$</th>
<th>$R_f$</th>
<th>$(LB, UB)$</th>
<th>Proposed VNS</th>
<th>GA</th>
<th>TS</th>
<th>hGA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$BC_{max}$</td>
<td>$AV(C_{max})$</td>
<td>$AV(CPU_s)$</td>
<td>$BC_{max}$</td>
</tr>
<tr>
<td>MK01</td>
<td>10 $\times$ 6</td>
<td>2.09</td>
<td>0.348</td>
<td>(36,42)</td>
<td>40</td>
<td>40</td>
<td>6.42</td>
<td>40</td>
</tr>
<tr>
<td>MK02</td>
<td>10 $\times$ 6</td>
<td>4.10</td>
<td>0.683</td>
<td>(24,32)</td>
<td>26</td>
<td>26.2</td>
<td>5.56</td>
<td>26</td>
</tr>
<tr>
<td>MK03</td>
<td>15 $\times$ 8</td>
<td>3.01</td>
<td>0.376</td>
<td>(204,211)</td>
<td>204</td>
<td>204</td>
<td>28.78</td>
<td>204</td>
</tr>
<tr>
<td>MK04</td>
<td>15 $\times$ 8</td>
<td>1.91</td>
<td>0.238</td>
<td>(48,81)</td>
<td>60</td>
<td>60.2</td>
<td>12.64</td>
<td>60</td>
</tr>
<tr>
<td>MK05</td>
<td>15 $\times$ 4</td>
<td>1.71</td>
<td>0.427</td>
<td>(168,186)</td>
<td>173</td>
<td>173</td>
<td>15.43</td>
<td>173</td>
</tr>
<tr>
<td>MK06</td>
<td>10 $\times$ 15</td>
<td>3.27</td>
<td>0.218</td>
<td>(33,36)</td>
<td>59</td>
<td>60</td>
<td>36.92</td>
<td>63</td>
</tr>
<tr>
<td>MK07</td>
<td>20 $\times$ 5</td>
<td>2.83</td>
<td>0.566</td>
<td>(133,157)</td>
<td>140</td>
<td>140.8</td>
<td>10.62</td>
<td>139</td>
</tr>
<tr>
<td>MK08</td>
<td>20 $\times$ 10</td>
<td>1.43</td>
<td>0.143</td>
<td>(523,523)</td>
<td>523</td>
<td>523</td>
<td>41.38</td>
<td>523</td>
</tr>
<tr>
<td>MK09</td>
<td>20 $\times$ 10</td>
<td>2.53</td>
<td>0.253</td>
<td>(299,369)</td>
<td>307</td>
<td>307.8</td>
<td>52.23</td>
<td>311</td>
</tr>
<tr>
<td>MK10</td>
<td>20 $\times$ 15</td>
<td>2.98</td>
<td>0.198</td>
<td>(165,296)</td>
<td>207</td>
<td>208.4</td>
<td>59.74</td>
<td>212</td>
</tr>
<tr>
<td>Average improvement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+0.93</td>
<td>-0.35</td>
<td>-0.80</td>
<td></td>
</tr>
</tbody>
</table>
where $BC_{\text{max}}$ is the best makespan obtained by the reported algorithm and $LB$ is the best-known lower bound. The results of Table 6 showed that in optimisation of those four groups of problems our algorithm outperformed two GAs, regarding MRE criterion. But hGA and TS algorithms worked better than the proposed VNS algorithm in respect of this criterion.

The results of the Hurink Vdata problems revealed the appropriate performance of our algorithm in optimising problems with a high flexibility. For instance, Figure 7 depicts the diagram of the algorithm convergence to minimise makespan for problem la18 of Vdata of HUdata. The best makespan which equals to 663 is reached after 44 iterations and is optimal.

Regarding the results obtained from the computational study, it seems that the proposed algorithm can be an effective approach for FJSP.

8. Conclusions and future study
In this paper, we developed a variable neighbourhood search algorithm for the flexible job-shop scheduling problem. Minimisation of the makespan was considered as the
objective function. A task sequencing list was used to represent the solutions and a combination of strategies was utilised for generating the initial solution. In the presented optimisation method, the external loop controlled the stop condition of the algorithm and the internal loop executed the search process. To search the solution space, the internal loop used two main search engines of VNS, i.e., the shake and local search procedures. In addition, various neighbourhood structures in relation to assignment and sequencing problems were employed in the shake and local search procedures of the presented algorithm. The proposed algorithm was tested on 181 benchmark problems. An extensive computational study revealed that the results obtained from our algorithm were quite analogous to those of famous algorithms. In the following, suggestions are offered for future work:

- Implementing effective parallelisation strategies in the VNS algorithm to improve the solution quality in FJSP.
- Developing hybrid algorithms such as VNS-SA and VNS-PSO for FJSP.
- Considering other constraints such as random processing time of the operations on the machines, and precedence constraints among operations of different jobs.
- Applying the VNS algorithm in a multi-objective FJSP.
- Developing efficiently intelligent neighbourhood structures for better and more diverse searches in the solution space of FJSP.

References


