International Journal of Business, Human and Social Sciences
ISSN: 2517-9411
Vol:9, No:8, 2015

A General Variable Neighborhood Search Algorithm to Minimize Makespan of the Distributed Permutation Flowshop Scheduling Problem

G. M. Komaki, S. Mobin, E. Teymourian, S. Sheikh

***Abstract*—**This paper addresses minimizing the makespan of the distributed permutation flow shop scheduling problem. In this problem, there are several parallel identical factories or flowshops each with series of similar machines. Each job should be allocated to one of the factories and all of the operations of the jobs should be performed in the allocated factory. This problem has recently gained attention and due to NP-Hard nature of the problem, metaheuristic algorithms have been proposed to tackle it. Majority of the proposed algorithms require large computational time which is the main drawback. In this study, a general variable neighborhood search algorithm (GVNS) is proposed where several time-saving schemes have been incorporated into it. Also, the GVNS uses the sophisticated method to change the shaking procedure or perturbation depending on the progress of the incumbent solution to prevent stagnation of the search. The performance of the proposed algorithm is compared to the state-of-the-art algorithms based on standard benchmark instances.

***Keywords*—**Distributed permutation flow shop, scheduling, makespan, general variable neighborhood search algorithm.

I. Introduction

THIS paper addresses the distributed permutation flow shop scheduling problem (hereafter DPFSP) which is extension of the classic permutation flow shop scheduling problem (PFSP). Since PFSP was a good approximation of production systems, it has been studied for decades, for comprehensive survey see [1]-[3], but nowadays due to globalization and competitive situations, managers have to switch from centralized production system to geographically distributed systems to reduce production costs and increasing flexibility to meet abrupt market changes. Each of the factories of the distributed production system has to perform several tasks subject to different constraints such as labor costs, local regulations, and tax and trading policies. Hence, managing the distributed production is more complex than single site production site.

G. M. Komaki is with the Electrical Engineering and Computer Science Department, Case Western Reserve university, Cleveland, OH 44106 USA (phone: 216-368-4114; e-mail: gxk152@case.edu).

M. Mobin is with the Department of Industrial Engineering and Engineering Management, Western New England University, Springfield, MA 01119 USA (Phone: 413-801-7845; e-mail: mm337076@wne.edu)

E. Teymourian is with the School of Mechanical, Industrial, and Manufacturing Engineering, Oregon State University, Corvallis, OR 97331 USA (e-mail: ehsan.teymorian@gmail.com).

S. Sheikh is with the Department of Management Science at New York Institute of Technology, New York, NY 10023 USA (e-mail: shaya.sheikh@case.edu).

One of the important classes of the distributed production systems with practical implication is the DPFSP, introduced by [4]. In this system, there are identical parallel factories (flow shops) each with serial of machines. Each job should be allocated to one of these factories and all of the operations of the job must be performed without preemption in the allocated factory. The goal is finding the allocation of jobs to the factories and the sequence of jobs of allocated to each factory such that makespan of the latest factory is minimized. Since minimizing the makespan of the PFSP is NP-hard, minimizing the makespan of the DPFSP due to more constraints is NP-Hard [4].

Naderi and Ruiz [4] investigated performance of the several mathematical models of the DPFSP. Also, they proposed several heuristic algorithms including SPT, LPT, and NEH which assign an index to each job and then to assign each job to factories two rules are suggested; (1) assign the job to the factory with the lowest current makespan and (2) assign the job to the factory that yields the minimum makespan after assigning the job. Based on extensive experiments, they have concluded that NEH using second rule, called NEH2, has the best performance among the other developed heuristic algorithms. They also proposed two variable neighborhood search algorithms which both use NEH2 to generate the initial sequence, but they are differ in term of acceptance criteria, the first version called VND(a) accepts the new solution if it improves the makespan of the current solution and the other version, called VND(b) accepts the new solution if it improves makespan of any factory. Based on experiments, they reported that VND(a) outperforms the VND(b).

Later, this problem has been investigated by [22] and they proposed Tabu Search (TS) algorithm. Based on experiments, they concluded that TS outperforms VND(a) [4]. Lin et al. [23] proposed algorithms based on iterated greedy (IG) algorithm and the best one among them named IG_{vst} outperforms VND(a). Recently, [5] proposed scatter search (SS) algorithm and it outperforms the existing algorithms. Xu et al. [25] also addressed the DPFSP and proposed hybrid immune algorithm (HIA) and reported several improved upper bounds for the standard benchmark instances. Recently, Fernandez-Viagas and Framinan [19] investigated the same problem and proposed iterated search algorithm named BSIG which the search is bounded by simple and effective rule. They compared to TS [22], EDA [24], and IG_{vst} [23] and concluded that BSIG outperforms them.

Since the DPFSP is an extension of the classical PFSP,

properties of the PFSP, for instance, properties developed by [26], and [11] can be applied to reduce the computational time and increase the efficiency of the proposed algorithms. Tillard's acceleration method to compute makespan has been applied to the DPFSP by [4] but block properties for the classical PFSP proposed by [8], [11] has not been investigated in the DPFSP. Using these properties, the search process can be limited only to movements that can improve the current solution; hence, it reduces the neighborhood size greatly without deteriorating the search power of the algorithm. Also, all of the proposed algorithms so far for the DPFSP use the same neighborhood structure (and local search) to perturb the current incumbent solution regardless of progress of the algorithm, in another words, when using one type of neighborhood structure for several iterations of the algorithm cannot improve the current incumbent solution, it indicates that the algorithm has trapped in the local solution if it is not the global solution. The former case is more probable since the addressed problem has many local optima solutions. In this situation, trapping in the local optima, using the same perturbation scheme it is unlikely to help the algorithm to escape from the local optima; therefore, it should use stronger perturbation. If the algorithm improves the current incumbent solution, the perturbation scheme should use low level scheme to give chance to investigate the neighborhood of the current solution. In this paper, a general variable neighborhood search (GVNS) algorithm is proposed which utilizes both of the above mentioned concepts, i.e., it uses the perturbation scheme depending on progress of the incumbent solution, also, the time saving strategies based on block properties of the PFSP are extended to the DPFSP.

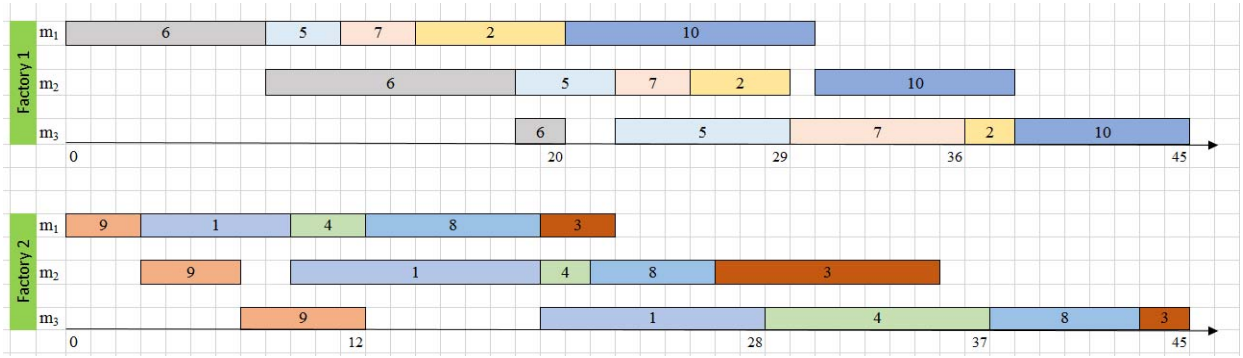


Fig. 1 Gantt chart of the numerical example

TABLE I
PROCESSING TIME OF JOBS

job	1	2	3	4	5	6	7	8	9	10
Stage 1	6	6	3	3	3	8	3	7	3	10
Stage 2	10	4	9	2	4	10	3	5	4	8
Stage 3	9	2	2	9	7	2	7	6	5	7

The rest of the paper is organized as follows. Next section is devoted to the problem description, and In Section III, properties of the DPFSP are investigated. In Section IV, the proposed general variable neighborhood search algorithm is discussed; and its performance is discussed in Section V. Finally, Section VI is devoted to conclusion and future studies.

II. PROBLEM DESCRIPTION

This paper deals with the distributed permutation flowshop scheduling problem where there are F identical parallel flowshops (factories) each with m serial machines. There is a set of jobs $J=\{1,2,..,n\}$ and set of machines $M=\{1,2,...,m\}$ and job j, $j \in J$, has m operations that should be allocated to one the factories and all operations of the job should be performed sequentially in the allocated factory. The other assumptions are as following. Each machine at a time can process only one job and each job at a time can be processed only by one machine. Pre-emption is not allowed, i.e., after processing of a job is started it cannot be stopped. The sequence of jobs in a factory for all machines is the same. All operations of jobs should be performed in the allocated factory.

The goal is finding assignment (allocation) of jobs to each flowshop and their sequence such that the makespan of the system, the completion time of the latest factory, is minimized. Consider the following notations:

- F: the number of factories
- n: the number of jobs
- m: the number of machines of each factory f, $f=1,..,F$.
- p_{ij}: processing time of job j at stage (or machine) i, $j \in \{1,..,n\}$, $i \in \{1,..,m\}$
- C_{ij}: completion time job j on machine i,

Let $\pi=(\pi^1, \pi^2,\ldots,\pi^f,\ldots,\pi^F)$ where $\pi^f=(\pi^f(1),\pi^f(2),\ldots,\pi^f(n_f))$ represents the sequence of jobs allocated to factory f, $1 \le f \le F$, and n_f is the number of the allocated jobs to the factory f. It is obvious that $\sum_{f=1}^{F} n_f = n$. Then, makespan, C_{max}, can be calculated by:

$$C_{max}(\pi^f) = \max_{1\le t_1 \le t_2 \le \cdots \le t_{m-1} \le n_f} \left(\sum_{j=1}^{t_1} p_{1\pi^f(j)} + \sum_{j=t_1}^{t_2} p_{2\pi^f(j)} + \cdots + \sum_{j=t_{m-1}}^{n_f} p_{m\pi^f(j)} \right) \quad (1)$$

and

$$C_{max}(\pi) = \max_{1\le f \le F} \left(C_{max}(\pi^f) \right) \quad (2)$$

where (1) calculates the makespan of factory f and (2)

represents the makespan of sequence π.

To be clear, consider a numerical example with 10 jobs and 2 factories each with 3 machines where the processing time of jobs is given in Table I. Assume that π^1=(6-5-7-2-10) (sequence of jobs in the first factory) and π^2=(9-1-4-8-3) (sequence of jobs in the second factory). The Gantt chart is presented in Fig. 1, and makespan of both factories is 45, therefore, makespan of the system is 45.

III. Properties of the DPFSP

In the following, two properties of the DPFSP developed by [19] are discussed.

Property 1[19]**.** Let π= (π^1,π^2,…,π^f,…,π^F) be the current jobs sequence of the factories. Adding job j* to the current solution of factory f, π^f, will increase the $C_{max}(\pi^f)$ at least by $\min_i(p_{ij*})$, i.e., $C_{max}(\pi'^f) \geq C_{max}(\pi^f)+\min_i(p_{ij*})$.

Property 2[19]**.** Adding job j* to the current solution of factory *f*, π^f, will increase the $C_{max}(\pi^f)$ at most by $\sum_{i=1}^{m} p_{ij*}$, i.e., $C_{max}(\pi'^f) \leq C_{max}(\pi^f)+\sum_{i=1}^{m} p_{ij*}$.

Properties 1 and 2 provide lower bound and upper bound of makespan of the factory *f* by adding a new job to its current partial solution, respectively.

Efficiency of algorithms highly depends on their search ability. The total number of possible solutions of the DPFSP is $\binom{n-1}{F-1} n!$ [4], which sharply increases as size of the problem increases. Since the DPFSP is an extension of the well-studied permutation flowshop scheduling problem, the well-stablished properties of the flowshop problem can be benefited to tackle the DPFSP and reduce the search space. For instance, [26] experimentally showed the insertion-based movements outperform the swap-based movements. Also, [11] showed that all insertion movements do not yield a solution with better makespan than current solution, in other words, the movements inside the blocks do not improve the makespan. Furthermore, they experimentally showed that movements strongly depend on the structure of blocks, i.e., it is enough to examine the movements to the immediately preceding and succeeding blocks. In the following, the block properties of the PFSP and its extension to the DPFSP are discussed.

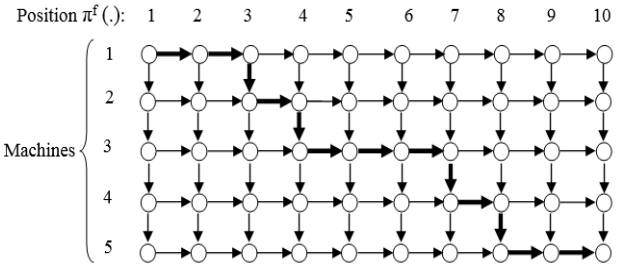


Fig. 2 Grid graph of factory f of the DPFSP (modified from [6], [11])

Recall (1) which calculates the makespan of the π^f, $1 \leq f \leq F$. Each factory of the DPFSP can be presented by a grid graph where the length of the longest path in the grid graph represents the makespan of the factory. This is extension of the grid graph presented by [11]. Fig. 2 shows a hypothetical factory with 5 machines and 10 jobs and critical path is shown by thin arrows.

Any path in the grid from node (1,1) to (m,n_f) can be represented by sequence of integers $t^f = (t_0^f, t_1^f, \dots, t_{m-1}^f, t_m^f)$ where t_0^f=1 and t_m^f=n_f and the makespan of the factory *f* can be presented as:

$$C_{max}(\pi^f) = \max_{1 \leq t_1^f \leq \cdots \leq t_{m-1}^f \leq n_f} \left(\sum_{j=1}^{t_1^f} p_{1\pi^f(j)} + \sum_{j=t_1^f}^{t_2^f} p_{2\pi^f(j)} + \cdots + \sum_{j=t_{m-1}^f}^{n_f} p_{m\pi^f(j)} \right) \quad (3)$$

Another equivalent makespan formulation can be presented as [6]:

$$C_{max}(\pi^f) = \max_{1 \leq q_1^f \leq \cdots \leq q_{n_f-1}^f \leq m} \left(\sum_{i=1}^{q_1^f} p_{i\pi^f(1)} + \sum_{j=q_1^f}^{q_2^f} p_{i\pi^f(2)} + \cdots + \sum_{j=q_{n_f-1}^f}^{m} p_{i\pi^f(n_f)} \right) \quad (4)$$

where sequence of integers $q^f = (q_0^f, q_1^f, \dots, q_{n_f-1}^f, q_{n_f}^f)$ represents the same path represented by t^f in (3).

As presented in Fig. 2, the path has horizontal sub-paths as well as vertical sub-paths; each of these sub-paths represents a block. Each path in the grid has *m horizontal blocks* (HB) and n_f *vertical blocks* (VB). Note that (3) and (4) calculate the makespan based on HBs and VBs, respectively.

Let path u^f =($u_0^f, u_1^f, \dots, u_{m-1}^f, u_m^f$) be the critical path in (3), then horizontal blocks can be defined as HB_l^f = ($\pi^f(u_{l-1}^f)$, $\pi^f(u_{l-1}^f+1)$,…, $\pi^f(u_l^f)$) where it represents *l*th *horizontal block* of factory *f,* $1 \leq l \leq m$ and $1 \leq f \leq F$, where $\pi^f(u_{l-1}^f)$ and $\pi^f(u_l^f)$ present the first and last jobs in HB_l^f. Also, the last job of HB_l^f is the first job of HB_{l+1}^f, l=1,2,…,m-1. The number of jobs of HB_l^f is $|HB_l^f|$= u_l^f-u_{l-1}^f+1. Consider Fig. 2, the horizontal blocks presented in this figure can be presented by u^f =(1,3,4 ,7,8,10).

Similarly, let v^f =($v_0^f, v_1^f, \dots, v_{n_f-1}^f, v_{n_f}^f$) be the critical path in (4), then the *r*th vertical block of the factory *f* can be represented as $VB_r^f = (v_{r-1}^f, v_{1-1}^f + 1, \dots, v_r^f)$ where $v_0^f = 1$, $v_{n_f}^f = m$, $1 \leq r \leq n_f$, and $1 \leq f \leq F$. the vertical blocks of the grid graph presented in Fig. 2 can be presented by v^f = (1,1,2 ,3,3,3,4,5,5,5).

Several researchers such as [8], [11], [7], and [6] investigated the properties of the blocks in the PFSP and they have designed effective algorithms based on these properties. In the following, some of the properties that have been employed in this research is presented. Since computing the change of makespan using the vertical blocks is simpler, the properties are presented based on the vertical blocks.

Property 3. Let v(a,b) present insertion movement where the job in position *a* of factory *f* is removed and inserted in position *b*. The change of makespan is [6]:

$$\Delta_{v(a,b)}=\begin{cases}\tau_1 & if\ a<b\\ \tau_2 & if\ a>b\end{cases} \quad (5)$$

where

$$\tau_1=\sum_{k\in VB_a}(p_{k\pi^f(a+1)}-p_{k\pi^f(a)})+\sum_{k\in VB_b}(p_{k\pi^f(a)}-p_{\pi^f(b)k})+ p_{v^f_{b-1}\pi^f(b)}-p_{v^f_a\pi^f(a+1)} \quad (6)$$

and

$$\tau_2=\sum_{k\in VB_a}(p_{k\pi^f(a-1)}-p_{k\pi^f(a)})+\sum_{k\in VB_b}(p_{k\pi^f(a)}-p_{\pi^f(b)k})+ p_{v^f_b\pi^f(b)}-p_{v^f_{a-1}\pi^f(a-1)} \quad (7)$$

The complexity of insertion movements is $O(n_f)$ [6], according to this property, if $\Delta_{v(a,b)}<0$, the movement is promising, otherwise it doesn't worth to investigate the movement. Therefore, this property can save the computational time of the algorithm. Nowicki and Smutnicki [11], Grabowski and Wodecki [7] and Solimanpur et al. [9] investigated this neighborhood structure and proposed strategies that reduce the search space further without deteriorating the search ability of the algorithm. These schemes are discussed in Section IV.C.

Property 4. Let $v(a,b,c)$ present insertion movement where three jobs in positions a, b, and c are chosen from factory f where $a<b<c$ or $a>b>c$ and the job in position a is placed into the position b and the job in the position b is placed into the position c and finally, the job in the position c is placed into the position a. The change of makespan of factory f is [6]:

$$\Delta_{v(a,b,c)}=\sum_{k\in VB_a}(p_{\pi^f(c)k}-p_{\pi^f(a)k})+\sum_{k\in VB_b}(p_{\pi^f(a)k}-p_{\pi^f(b)k})+\sum_{k\in VB_c}(p_{\pi^f(b)k}-p_{\pi^f(c)k}) \quad (8)$$

The complexity of this neighborhood structure is $O(n_f^3)$ and checking all of them require tremendous time, but using this property only promising movements $\Delta_{v(a,b,c)}<0$ needs to be investigated.

IV. The Proposed Fast General Variable Neighborhood Search Algorithm

Mladenovic and Hansen [16] introduced a simple but powerful metaheuristic algorithm called Variable Neighborhood Search (VNS) based on idea of systematic changing of neighborhood search algorithms. Since then, the VNS has been applied to solve many combinatorial problems including traveling salesman problem, scheduling problems and so on. For a comprehensive review, readers are referred to [18] and [17].

The VNS algorithm has two main features; shaking procedure and local search algorithm. The shaking procedure is a perturbation which helps the algorithm to escape from current local optima and local search algorithm is responsible to search the neighborhood of the current solution based on the predefined neighborhood structures. Serval extensions of the original VNS by changing one or two of these features have been proposed including Variable Neighborhood Descend (VND), Reduced VNS (RVNS), and General VNS (GVNS) [15] and so on. Main difference of these extensions is based on their exploration methodology or neighborhood search algorithms which could be deterministic, random, or mixed (combination of both random and deterministic) [15].

GVNS

Initialization: (*x*: *Initial solution*)
1. **While** $t<t_{max}$
2. $k\leftarrow 1$
3. While $k<k_{max}$
4. $x'\leftarrow Shake(x,k)$
5. $x''\leftarrow VND(x')$
6. If x'' is better than x, set $k\leftarrow k+1$
7. Else $k\leftarrow 1$
8. EndIf
9. EndWhile
10. t← CPU time
11. **EndWhile**

Fig. 3 Pseudo-code of the GVNS [15]

Proposed GVNS for the DPFSP

Initialization:
1. π← Create initial solution using NEH2 % *Section IV.A*%
2. $\pi_{best}\leftarrow\pi$
3. Set *it* = 0, and diversification level *flag*=0
4. **While** $it < It_{max}$
5. π'←Apply *Shake*(π,*flag*) % *Section IV.B*%
6. k=1
7. While $k<k_{max}$
8. π''←Apply Local Search based on $N_k(\pi')$% *Section IV.C*%
9. if $C_{max}(\pi'')< C_{max}(\pi')$
10. $k\leftarrow 1$
11. $\pi'\leftarrow\pi''$
12. if $C_{max}(\pi')< C_{max}(\pi_{best})$
13. $\pi_{best}\leftarrow\pi''$
14. endif
15. Elseif
16. $k\leftarrow k+1$
17. Endif
18. EndWhile
19. If $C_{max}(\pi')< C_{max}(\pi_{best})$
20. $\pi\leftarrow\pi'$
21. *flag=1*
22. Else
23. $\pi\leftarrow\pi_{best}$
24. *flag= flag+1*
25. EndIf
26. $it\leftarrow it+1$;
27. **EndWhile**
28. Report π

Fig. 4 Pseudo-code of the proposed GVNS algorithm

The VND uses deterministic local search algorithm, the RVNS is based on the random one, and the GVNS uses the mixed of deterministic and random methodologies. The random search methodology has higher diversification ability while the deterministic search has more emphasize on the intensification. Since the GVNS as well as the canonical VNS use both random and deterministic search, the algorithms have balanced intensification and diversification search [15], therefore, they have higher chance to find the global optima. The pseudo-code of the GVNS is presented in Fig. 3. The main difference of the canonical VNS and GVNS is that the first one uses local search to improve the perturbed solution

while the former one uses the VND.

In this study, the GVNS is applied to tackle the DPFSP which some improvements are incorporated into the proposed algorithm based on properties of the DPFSP, see Fig. 4. The proposed GVNS starts with an initial solution generated by the NEH2 [4] and generates a new solution using *shake* procedure. Majority of studies have suggested the same shake procedure at each iteration of the algorithm, for instance [21] and [4]. Intuitively, when the algorithm is trapped in the local optima, a stronger perturbation is needed to escape from the local optima [12]-[14], if it was successful, then the algorithm should focus on intensification, exploring the neighborhood of the current solution to find the (local) optimal, otherwise another stronger perturbation needs to be applied until the global optimal is found. This concept has been applied to VNS by [15] and [13], [14]. In this study, the GVNS is based on the same concept, i.e., the severity of the shake procedure depends on the statues of the current solution and it is determined by parameter *flag*. If *flag*=0, it represents that the shake procedure will not be applied, but if the outer iteration of the algorithm, lines 4-27, was unsuccessful for a predefined number of iterations to improve the current incumbent solution, π_{best}, an stronger perturbation procedure will be applied. The local search algorithm uses the neighborhood search based on insertion which is simple and fast neighborhood, and as the number of the iteration, k, gets closer to the iteration limit (k_{max}) of the inner loop, lines 7-18, it uses more complicated neighborhood search.

In the following subsections, the initial solution, shaking procedure and local search algorithm are discussed.

A. The Initial Solution

Several researches have pointed out the importance of the effect of the initial solution of the metaheuristic algorithms on the quality of the algorithm. As pointed out earlier, NEH2 [4] outperforms the developed constructive heuristic algorithms, hence, this algorithm is used to generate the initial solution of the proposed GVNS. In Fig. 5, steps of NEH2 are presented.

NEH2

1. Compute $t_j = \sum_{i=1}^{m} p_{ij}$ for j=1,2,...,n
2. Sort t_j in non-decreasing order, let S be the sequence of jobs and set $\pi^f = \phi$ for f=1,2,...,F
3. **For** f=1:F
4. Assign $j_s = S(1)$ to factory f, i.e., $\pi^f = \{j_s\}$, $S \leftarrow S \backslash j_s$
5. **EndFor**
6. **For** j=1: n-F
7. **For** f=1:F
8. **For** k=1:n_f
9. Insert job $S(j)$ into position k of factory f
10. **EndFor**
11. **EndFor**
12. Select the best sequence
13. **EndFor**

Fig. 5 Pseudo code of NEH2

B. Shaking Procedure

The ultimate goal of the shaking procedure is finding the better neighborhood of the current solution. Most of studies use the same shaking procedure at any iteration of the algorithm. This may lead the algorithm to the local optima. Here, different shaking procedures depending on the statues of the algorithm is used, i.e., if an iteration of the algorithm was successful in term of improving the current incumbent solution, the algorithm uses low level shaking procedure to lead the solution to the better neighborhood of the current solution, if the algorithm was not successful then another stronger shaking procedure is used to help the algorithm escape from the current local optima. This idea has been suggested by [15] and applied to minimize total tardiness of the classical PFSP by [13], [14] and the result was astonishing.

In this study, the following shaking procedures are defined.

- ***Shake*(π,1):** This shaking procedure is low level shake and it is applied on factory or factories with the latest makespan. Here, we suggest 2-Opt operator in which two jobs from *two different (horizontal) blocks* are selected and their position is changed and the sequence of jobs between these two jobs is reversed. If the two jobs are from the same horizontal block, the shaking procedure is not strong enough since there is high chance to be undone by the local search algorithm presented in the next subsection.
- ***Shake*(π,2):** Since the algorithm for an iteration (using all possible movements defined) was unsuccessful to improve the current incumbent solution, a higher shaking level needs to be applied. In this procedure, the jobs from the factory with the latest completion time is randomly chosen and inserted into the sequence of the factory with the earliest completion time. In case of tie, i.e., there is more than one factory with the latest (or earliest) completion time, chose randomly.
- ***Shake*(π,*flag*≥3):** Since the algorithm was not able to find the better neighborhood in the previous iterations, extremely strong shaking procedure is needed to be applied. Here, cyclic exchange is suggested where some jobs from each factory are randomly selected and inserted in other factory. In other words, the selected jobs from factory 1 are randomly inserted to the factory 2, and jobs picked up from factory 2 are inserted in factory 3, as so on. Finally, the jobs selected from factory F is inserted in factory 1.

It is well-established that in scheduling problems the majority of time of any algorithm is spent on calculating the objective function, for instance [11] reported that 80-95% of their proposed algorithm is consumed to calculate the makespan of the neighborhood of the current solution. Hence, to design more efficient and fast algorithm one needs to use acceleration methods to reduce the computational time. There are two possible options; the first one is limiting the neighborhood search to the most promising movements, as discussed in the next section, the second strategy is accelerating the calculation of the makespan after any change on the current solution. In this study, the *advanced implementation* of [11] is used. Since *Shake*(π,1) is applied on the factory f^{max}, and sequence of jobs of the other factories remains unchanged, one need to update the completion time of

jobs affected by the shaking procedure. Similarly, for other shaking procedures, only the completion times of the affected jobs in each factory should be considered. This simple strategy can save a lot of time.

C. Local Search Algorithm

Local search algorithm is responsible to search the feasible area to find the better solution than the current solution. The basic concept of the VND (and VNS) is using local search algorithm with different neighborhood structures which switches systematically from one to another neighborhood structure. This strategy helps the algorithm to escape from local optima. When the local search algorithm with a neighborhood structure is unable to find the better solution, to prevent stagnation of the search procedure the algorithm should apply the stronger neighborhood structure.

Similar to the shaking procedure explained in the previous subsection, the local search algorithm starts with a simple neighborhood structure and improves the current incumbent solution until it not further improvement cannot be found, then another stronger neighborhood structure is applied and it improves the current incumbent solution until it cannot improve it further. This procedure continues until all predefined neighborhood structures have been tried as presented in Fig. 4, lines 6-18.

Due to the structure of the DPFSP, two types of neighborhood structures are needed to be defined; the neighborhood that changes the sequence of the jobs of a factory (with the latest makespan) and another one that changes jobs across factories. The proposed local search algorithms for the DPFSP are mainly based on swap and insertion within a factory and also across factories, for instance, see [4], [5]. The swap movement across factories is based on selection of two factories which one of them has the latest makespan and then the randomly selected jobs are swapped. In the insertion movement, one job is selected from the factory with the latest makespan and inserted in a random position of jobs sequence of another factory.

In this study, the following neighborhood structures are considered.

- ***$N_1(\pi)$*:** This neighborhood structure is based on insertion of a job on the factory or factories with the latest completion time, i.e., the job in position *a* is randomly chosen and inserted in position *b* that gives the best possible sequence. This neighborhood structure has been applied by [4] and [5] which use full search meaning that all possible movements are tried while it is not necessary. Nowicki and Smutnicki [11] experimentally showed movements inside of (horizontal) blocks do not improve the makespan. Furthermore, [9] and [7] concluded that it is enough to investigate only the immediate preceding and following blocks of a block of jobs. In other words, for each job there are two possible positions to insert, right after the last job of its block and right before the first job of its block, see Fig. 6 which shows the possible movements based on the $N_1(\pi)$ neighborhood structure where the dark circles represent the boundary of block and white circles represent the jobs inside each block. Note that the jobs inside of the first block and the last block are not allowed to perform leftward moves and rightward moves, respectively.

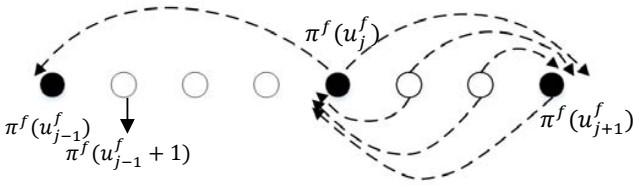


Fig. 6 Possible movements defined by $N_1(\pi)$

TABLE II
AVERAGE RPD OF THE ALGORITHMS BASED ON N×M

n×m	GVNS	BSIG[1]	TS	EDA	IG_{vst}	VND(a)
20×5	0.1531	0.1064	3.9570	0.9045	0.2919	5.3859
20×10	0.1864	0.0880	3.0521	0.8364	0.1669	4.0749
20×20	0.0771	0.0553	2.1839	0.6602	0.1284	2.7800
50×5	**0.4130**	0.4263	3.0883	3.1043	0.8581	4.1931
50×10	0.8962	0.8769	4.1410	3.3479	1.3246	5.3762
50×20	0.9338	0.8642	3.6033	2.8207	1.1563	4.6535
100×5	0.3104	0.2568	1.3438	3.3870	0.5418	2.3327
100×10	0.8344	0.7141	2.4741	3.9441	1.0511	3.7668
100×20	**0.7375**	0.8804	2.5502	2.9112	1.1105	3.8512
200×10	0.6777	0.6367	1.5329	2.7425	0.8481	2.7675
200×20	0.8935	0.8622	1.8742	4.9287	0.9587	3.1120
500×20	**1.0310**	1.0687	1.8494	7.0186	1.2804	2.6916
Ave.	0.5953	**0.5697**	2.6375	3.0505	0.8097	3.7488

1- Stopping criteria is $n \cdot m \cdot F \cdot 2$ms.

Recall property 3 presented in Section III, according to this property only those movements with $\Delta_{N_1(\pi)}<0$ need to be investigated. This helps to further reduce the search space and thus the computational time of the algorithm will reduce without deteriorating the search power of the algorithm.

- ***$N_2(\pi)$*:** In this neighborhood structure, three jobs in positions *a*, *b*, and *c* are randomly chosen from the factory with the latest makespan where $a<b<c$ or $a>b>c$ and the job in position *a* is placed into the position *b* and the job in the position *b* is placed into the position *c* and finally, the job in the position *c* is placed into the position *a*. The complexity of this neighborhood structure is $O(n_f^3)$. If all of the selected jobs are from the same block, then according to the property 3, it is not a promising movement, hence, the selected jobs should be from different (horizontal) blocks, or at least one of them should be from different (horizontal) block. In order to reduce the computational time required to examine all space of this neighborhood structure, the focus should be only on the promising movements which can be identified by the property 4 presented in Section III. According this property, only the promising movements, i.e., $\Delta_{N_2(\pi)}<0$ needs to be investigated which help to reduce the search space and thus the computational time of the algorithm will reduce.
- ***$N_3(\pi)$*:** In this neighborhood structure, a job in position *a*

of the factory f_1 with the latest makespan is selected and reinserted in a random position b of factory f_2 which is randomly selected.

The properties 1 and 2 presented in the Section III provide lower and upper bounds of inserting a job into current solution of the factory. Therefore, it can be incorporated to reduce the computational time of the search algorithm. Assume the makespan of current solution is C_{max}, and the job in position a of the factory f_1 is going to be inserted into sequence of factory f_2, if $\min_i(p_{i\pi^{f_1}(a)}) + C_{max}(\pi^{f_2}) \leq C_{max}$, then the movement is promising movement and can be tested, otherwise it doesn't worth to investigate the movement.

- **$N_4(\pi)$:** In this neighborhood structure, a job in position a of the factory f_1 with the latest makespan is selected and swapped with the job in position b of factory f_2.

Here, we develop the following property where compute the change of makespan of each factory as:

$$\Delta_{N_4}^{f_1} = \sum_{k \in VB_a^{f_1}} (p_{\pi^{f_2}(b)k} - p_{\pi^{f_1}(a)k}) \quad (9)$$

$$\Delta_{N_4}^{f_2} = \sum_{k \in VB_b^{f_2}} (p_{\pi^{f_1}(a)k} - p_{\pi^{f_2}(b)k}) \quad (10)$$

where $\Delta_{N_4}^{f_1}$ and $\Delta_{N_4}^{f_2}$ represent the change of the makespan of factory f_1 and f_2 due to apply $N_4(\pi)$, respectively. According to this property, if both $\Delta_{N_4}^{f_1}$ and $\Delta_{N_4}^{f_2}$ are positive then the movement is not promising, but if one of them is negative, it is considered as promising movement.

TABLE III
AVERAGE RPD OF THE ALGORITHMS BASED ON F

F	GVNS	BSIG	TS	EDA	IG_{vst}	VND(a)
2	0.8110	0.8082	2.1402	2.8234	0.9964	3.2591
3	0.8270	0.8124	2.5537	3.3619	1.0495	3.9059
4	0.6150	0.6018	2.6707	3.4470	0.8860	3.8746
5	0.4380	0.4107	2.7706	3.4973	0.6805	3.8897
6	0.4810	0.4350	2.8715	3.5977	0.6952	3.9428
7	0.3970	0.3500	2.8184	3.4907	0.5508	3.6207
Ave.	0.5948	0.5697	2.6375	3.3697	0.8097	3.7488

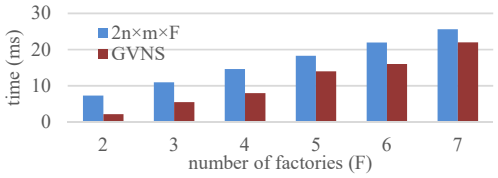
V. EXPERIMENTAL RESULTS

To evaluate the performance of the proposed GVNS, we conducted experiments based on available standard benchmark problems at http://soa.iti.es. In this set of problems, the number of jobs is n={20,50,100, 200, and 500} and the number of machines is m={5,10, and 20}, and the number of factories is F={2,3,4,5,6, and 7}, totally, there are 720 instances.

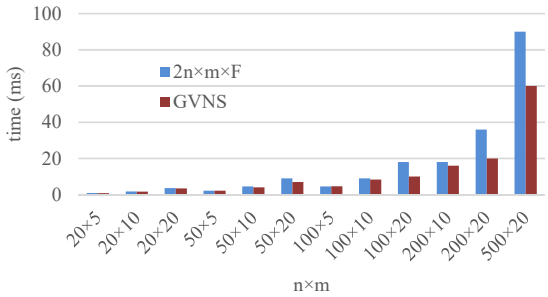
The proposed GVNS is compared to BSIG [10], TS [22], EDA [24], VND(a) [4] and IG_{vst} [23]. All algorithms are coded in C++ compiled with g++ (on awin7/64bit OS), and run on PC with Intel® Core™ i7-2600@ 3.4 GHz and 8 GB memory. The only parameter of the proposed GVNS is It_{max}, maximum iteration of the algorithm, and based on initial experiments it is set to 120. Due to random nature of the algorithm, the algorithm is run 5 times for each instance. The Relative Percentage Deviation (RPD) is measured as:

$$RPD = \frac{Alg_{Sol} - Best_{Sol}}{Best_{Sol}} \times 100 \quad (11)$$

where Alg_{Sol} and $Best_{Sol}$ represent the solution of any algorithm and the best known solution, respectively. The average RPD of the algorithms are presented in Tables II and III based on n×m and the number of factories (F), respectively. As can be observed, generally BSIG outperforms all algorithms including GVNS but there are cases that GVNS has better performance than BSIG, for instance for problems 50×5, 100×20, and 500×20 as bolded in Table II. In average, BSIG has the best performance, then GVNS, followed by IG_{vst}. This conclusion is verified by Table III as well as Fig. 8. In term of computational time, stopping criteria of all algorithms except the GVNS is $2\times n\times m\times F$ *ms* and the average computational time of the GVNS is based on number of factories is compared to the stopping criteria of other algorithms in Fig. 7. As can be observed, the computational time of the GVNS is much less than the other stopping condition while its performance is comparable to other algorithms.



(a) Based on F



(b) Based on n×m

Fig. 7 Comparing computational time of algorithms

VI. CONCLUSION

This paper addresses the distributed permutation flow shop scheduling problem (DPFSP) where there are identical parallel factories with series of machines. The goal is allocating jobs to the factories and finding the sequence of jobs allocated to each factory such that the completion time of the last processed job of the latest factory is minimized. Since the problem is NP-Hard, we proposed General Variable Neighborhood Search Algorithm (GVNS) which has two important features; (1) the shaking procedure depending on statues of the algorithm changes, if the algorithm has trapped in the local optima, a stronger shaking procedure is applied and if the algorithm is successful to improve the solution, a slight perturbation

scheme will be applied, (2) in the local search of the algorithm the search space is reduced to save computational time using properties of the addressed problem such as horizontal and vertical block properties, also an acceleration method to compute the makespan is incorporated.

The performance of the GVNS is compared to the state-of-the-art algorithms and it revealed the GVNS has very good performance and even in some instance its performance is better than all of them. In term of computational time, the proposed GVNS needs much less computational time.

The DPFSP recently gain attention and there are a lot of possible directions of research. In this study, we only consider the well-known neighborhood structures; one can investigate the other neighborhood structures to improve the performance of the proposed GVNS. Also, developing constructive metaheuristic algorithms such Ant Colony Optimization and Intelligent Water Drops algorithm due to great performance in solving combinatorial problems as reported in the literature can be an interesting direction of research.

As another direction of research, one can study the DPFSP with different criteria and different constraints, for instance one can consider the DPFSP with no-wait constraint and use the properties of the no-wait flow shop available in the literature such as [20].

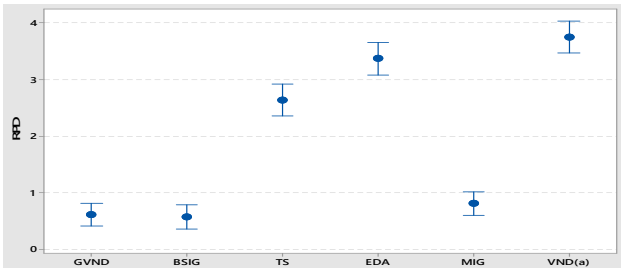


Fig. 8 Means plot and 95% confidence level

ACKNOWLEDGMENT

The first author thanks Professor Bahman Naderi for sharing the best solutions of the instances as well as Professor Victor Fernandez-Viagas for sharing the detail of his experiments.

REFERENCES

[1] J. M. Framinan, J. N. Gupta, R. Leisten, "A review and classification of heuristics for permutation flow-shop scheduling with makespan objective," *J. Oper. Res. Soc*., vol. 55, no. 12, pp. 1243-1255, July 2004.
[2] S. R. Hejazi, S. Saghafian, "Flowshop-scheduling problems with makespan criterion: a review," *Int. J. Prod. Res.*, vol. 43, no. 14, pp. 2895-2929, 2005.
[3] J. N. Gupta, E.F. Stafford, "Flowshop scheduling research after five decades," *Eur. J. Oper. Res.,* vol. 169, no. 3, pp. 699-711, March 2006.
[4] B. Naderi, R. Ruiz, "The distributed permutation flowshop scheduling problem," *Comput. Oper. Res.*, vol. 37, no. 4, pp. 754–768, April 2010.
[5] B. Naderi, R. Ruiz, "A scatter search algorithm for the distributed permutation flowshop scheduling problem," *Eur. J. Oper. Res.*, vol. 239, no. 2, pp. 323-334, December 2014.
[6] B. Ekşioğlu, S. D. Ekşioğlu, P. Jain, "A tabu search algorithm for the flowshop scheduling problem with changing neighborhoods," *Comput. Ind. Eng.*, vol. 54, no. 1, pp. 1-11, February 2008.
[7] J. Grabowski, M. Wodecki, "A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion," *Comput. Oper. Res.,* vol. 31, no. 11, pp. 1891-1909, September 2004.
[8] E. Nowicki, C. Smutnicki, "Some aspects of scatter search in the flow-shop problem," *Eur. J. Oper. Res.*, vol. 169, no. 2, pp. 654-666, 2006
[9] M. Solimanpur, P. Vrat, R. Shankar, "A neuro-tabu search heuristic for the flow shop scheduling problem," *Comput. Oper. Res.*, vol. 31, no. 13,pp. 2151-2164, November 2004.
[10] V. Fernandez-Viagas, J. M. Framinan, "A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem," *Int. J. Prod. Res.*, vol. 53, no. 4, pp. 1111-1123, 2015.
[11] E. Nowicki, C. Smutnicki, "A fast tabu search algorithm for the permutation flow-shop problem," *Eur. J. Oper. Res.*, vol. 91, no. 1, pp. 160-175, September 1996.
[12] X. Dong, M. Nowak, P. Chen, Y. Lin, "Self-adaptive Perturbation and Multi-neighborhood Search for Iterated Local Search on the Permutation Flow Shop Problem," *Comput. Ind. Eng.,* vol. 87, pp. 176–185, September 2015.
[13] R. M'Hallah, "Minimizing total earliness and tardiness on a permutation flow shop using VNS and MIP," *Comput. Ind. Eng.*, vol. 75, pp. 142-156, September 2014.
[14] R. M'Hallah, "An iterated local search variable neighborhood descent hybrid heuristic for the total earliness tardiness permutation flow shop," *Int. J. Prod. Res.*, vol. 52, no. 13, pp. 3802-3819, 2014.
[15] J. Sánchez-Oro, J. J. Pantrigo, A. Duarte, "Combining intensification and diversification strategies in VNS. An application to the Vertex Separation problem," *Comput. Oper. Res.*, vol. 52, pp. 209-219, December 2014.
[16] N. Mladenovic, P. Hansen, Variable neighborhood search, *Comput. Oper. Res.*, 24 (1997) 1097–1100.
[17] P. Hansen, N. Mladenović, "Variable neighborhood search: Principles and applications," *Eur. J. Oper. Res.*, vol. 130, no. 3,pp. 449-467,2001.
[18] P. Hansen, N. Mladenović, J.A.M. Pérez, "Variable neighbourhood search: methods and applications," *4OR*, vol. 6, no. 4, pp. 319-360, 2008.
[19] V. Fernandez-Viagas, J. M. Framinan, "A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem," *Int. J. Prod. Res.*, vol. 53, no. 4, pp. 1111-1123, 2015.
[20] X. Li, Q. Wang, C. Wu, "Heuristic for no-wait flow shops with makespan minimization," *Int. J. Prod. Res.*, vol. 46, no. 9, pp. 2519-2530, 2008.
[21] R. Ruiz, T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *Eur. J. Oper. Res.*, vol. 177, no. 3, pp. 2033-2049, March 2007.
[22] J. Gao, R. Chen, W. Deng, "An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem," *Int. J. Prod. Res.*, vol. 51, no. 3, pp. 641-651, 2013.
[23] S. W. Lin, K. C. Ying, C. Y. Huang, "Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm," *Int. J. Prod. Res.*, vol. 51, no. 16, pp. 5029-5038, 2013.
[24] S. Y. Wang, L. Wang, M. Liu, Y. Xu, "An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem," *Int. J. Prod. Econ*., vol.145, no. 1, pp. 387-396, September 2013.
[25] Y. Xu, L. Wang, S. Wang, M. Liu, "An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem," *Eng. Optim*., vol. 46, no. 9, pp. 1269-1283, 2014.
[26] E. Taillard, "Some Efficient Heuristic Methods for the Flow Shop Sequencing Problem." *Eur. J. Oper. Res.,* vol. 47,no. 1, pp.65–74, 1990.