Minimizing the total completion time in a distributed two stage assembly system with setup times

Fuli Xiong, Keyi Xing, Feng Wang, Hang Lei, Libin Han

PII: S0305-0548(14)00031-8
DOI: http://dx.doi.org/10.1016/j.cor.2014.02.005
Reference: CAOR3504

To appear in: *Computers & Operations Research*

Minimizing the total completion time in a distributed two stage assembly system with setup times

Fuli Xiong[a,b], Keyi Xing[a*], Feng Wang[a], Hang Lei[a] and Libin Han[a]

[a]*State Key Laboratory for Manufacturing Systems Engineering, Xi'an Jiaotong University, Xi'an, Shanxi, 710049, China*

[b]*School of Information and Control Engineering, Xi'an University of Architecture and Technology, Xi'an, Shanxi, 710055, China*

Corresponding author. Email: kyxing@sei.xjtu.edu.cn (K. -Y. Xing)

Abstract

In this paper, a novel distributed two stage assembly flowshop scheduling problem (DTSAFSP) is addressed. The objective is to assign jobs to several factories and schedule the jobs in each factory with the minimum total completion time (*TCT*). In view of the NP-hardness of the DTSAFSP, we develop heuristics method to deal with the problem and propose three hybrid meta-heuristics (HVNS, HGA-RVNS, and HDDE-RVNS). The parameters of HGA-RVNS and HDDE-RVNS are tuned by using the Taguchi method and that of HVNS is done by using the single factor ANOVA method. Computational experiments have been conducted to compare the performances of the proposed algorithms. The analyses of computational results show that, for the instances with small numbers of jobs, HDDE-RVNS obtains better performances than HGA-RVNS and HVNS; whereas for the instances with large numbers of jobs, HGA-RVNS is the best one in all the proposed algorithms. Computational results indicate that the performances of the HDDE-RVNS and HGA-RVNS are not much affected by the number of machines at the first stage and factories. The experimental results also show that the RVNS-based local search steps in both HGA-RVNS and HDDE-RVNS are efficient and effective.

Keywords: scheduling; distributed two-stage assembly flowshop; hybrid meta-heuristics; total completion time

1. Introduction

The two-stage assembly flow shop problem (TSAFSP) can be considered as follows. There are *n* jobs to be processed. At the first stage, *m* different components of a job are processed on *m* parallel machines. When all of the components are completed at the first stage, a single assembly machine assembles the components together to finish processing the job. The TSAFSP has many industrial applications such as fire engine assembly plant [1], personal computer manufacturing [2], and distributed database systems [3], etc. It has received increasing attention over the past two decades.

Lee et al. [1] and Potts et al. [2] first addressed independently the TSAFSP with makespan criteria and both proved that the problem is NP-hard even for the case of $m = 2$ at stage one. A schedule may vary depending on the objective function of the TSAFSP. In the literatures, different objectives have been considered which help to find efficient schedules. For example, Lee et al. [1], Potts et al. [2], Koulamas and Kyparisis [4], and Allahverdi and Al-Anzi [6] considered the TSAFSP with makespan criterion. Allahverdi and Al-Anzi [3] and Al-Anzi and Allahverdi [5] addressed the problem with lateness criterion. Mozdgir et al. [7] addressed the problem under a weighed sum of makespan and mean completion time. Tozkapan et al. [8] considered the problem with objective of minimizing the total weighted flow time.

In the classical TSAFSPs, it is assumed that all jobs are produced in a single production system. However, in order to get lower production cost, decrease manufacturing period, and achieve higher product quality, many companies have changed their single factory production system to distributed production system which has multi-factories with similar functions [9, 10]. The distributed scheduling problems (DSPs) include two decision sub-problems: job allocation problem and job sequence problem. It is apparently that the problem is more complicated than the single factory scheduling problem. Recently, some works have been done on different DSPs such as distributed flowshop [9-12], distributed job shop [13-14], and distributed flexible manufacturing system (FMS) [15-16]. However, little work has been conducted on the DSP for the two stage assembly flowshop environment, which we called distributed two stage assembly flowshop scheduling problems (DTSAFSPs). In real situations, many scheduling problems can be modeled as distributed two stage assembly problems. For example, in a personal computer manufacturing company which have several sub-factories with similar functions. In each factory, central processing units, hard-disks, monitors, keyboards, and etc. are manufactured at the first stage, and all the required components are assembled to customer specification at a packaging station (the second stage). Another application is in a car engine company with several car engine factories. All parts or components for a car engine are produced in parallel at the first stage. Then, they are assembled into a car engine at the second stage. Under these circumstances, we study the DTSAFSP which is an extension of the two stage assembly scheduling problem (TSAFSP). To the best our knowledge, there are no published papers for dealing with the DTSAFSP. In the past decades, many objectives, such as makespan, total completion time, tardiness, of scheduling problems have been investigated. Total completion time indicates the time duration for each job to stay in the system. Thus, optimizing this objective leads to the minimization of in-process inventory, stable or even use of resources, and a rapid turn-around of jobs and improves customer service in terms of responsiveness,. Therefore, the TCT objective is very important in industry where reducing holding cost, inventory, and providing quick and good services to customers are of primary consideration [17-19]. It is a common assumption that setup time is included in processing times in production scheduling research. Although this assumption may be reasonable for some real scheduling problems, separate setup time must be considered in other situations, such as chemical, pharmaceutical, printing, food processing, metal processing, and semiconductor industries [20]. The benefit of considering separate setup times for our problem is that, in each factory, when there exists some idle time at the second stage (assembly stage), the setup time for assembly operations at the second stage can be performed prior to the maximum completion time of all the machines at the first stage. This indicates that the performance measure may be improved by considering setup times as separate from processing times. In this situation, this paper addresses the DTSAFSP to minimize the *TCT* (DTSAFSP-TCT) with sequence-independent setup times.

Recently, many meta-heuristic algorithms have been proposed to successfully solve combinational optimization problems which are strongly NP-hardness. The meta-heuristic algorithms include genetic algorithm (GA) [22], simulated annealing algorithm (SA) [23] , differential evolution (DE) [24], variable neighborhood search (VNS) [25], tabu search (TS) [26], ant colony optimization (ACO) [27], etc. Obviously, the DTSAFSP-TCT is strongly NP-hard since its special case (TSAFSP) is strongly NP-hard [17]. Therefore, in this paper we develop three hybrid meta-heuristics (HVNS, HDDE-RVNS and HGA-RVNS) for dealing with the problem.

The remainder of this paper is organized as follows. Section 2 describes the DTSAFSP-TCT under consideration and a property of the problem. The proposed algorithms including a SPT-based heuristic (ESPT) and three hybrid meta-heuristics (HVNS, HGA-RVNS and HDDE-RVNS) for the problem are presented in Section 3. Section 4 shows experimental results for the problem. Section 5 concludes the paper and gives suggestions for future research.

2. Problem definition and a property

At first, the pertinent notations for the DTSAFSP-TCT are introduced as follows.

***Notations*:**

n	total number of jobs;
m	total number of machines at the first stage of each factory;
f	total number of factories;
j, j'	job indices, $j, j' = 1, 2, \ldots, n$;
i, l	factory indices, $i, l = 1, 2, \ldots, f$;
k	machine indices at the first stage, $k = 1, 2, \ldots, m$;
$s_{j,k}$	setup time of job j on machine k at the first stage;
r_j	setup time of job j on assembly machine at the second stage;
$t_{j,k}$	processing time of job j on machine k at the first stage;
p_j	assembly time of job j on assembly machine at the second stage;
n_i	total number of jobs assigned to factory i;
π^i	sequence of jobs in factory i, $\pi^i = (\pi_1^i, \pi_2^i, \ldots, \pi_j^i, \ldots, \pi_{n_i}^i)$ where π_j^i is the job index in position j of factory i;
π	schedule for the DTSAFSP-TCT, $\pi = (\pi^1, \pi^2, \ldots, \pi^f)$;
$C_{j,k}$	completion time of job j on machine k at the first stage, $j = 1, 2, \ldots, n$; $k = 1, 2, \ldots, m$;
C_j	completion time of job j on assembly machine at the second stage, $j = 1, 2, \ldots, n$;
$C_{\max}(\pi)$	makespan, $C_{\max}(\pi) = \max_j \left(C_j\right)$;

$TCT(\pi)$ total completion time, $TCT(\pi) = \sum_{j=1}^{n} C_j$.

2.1. Problem definition

The problem studied here can be defined as follows. n independent jobs available at time zero from the set $J = \{1,2,\ldots,n\}$ need to be processed on f identical factories from the set $F = \{1, 2,\ldots, f\}$. All factories are identical and each factory has the same set M of m processing machines at the first stage and the same assembly machine at the second stage, just like in the classical TSAFSP with single factory. Each job can be produced in each factory. Assume that there are unlimited buffers between the first stage and the second stage in each factory. Each job can be processed on at most one machine at a time. Once a job is assigned to a factory, it cannot be transferred to the other factories. In each factory, when all m components of a job are processed at the first stage, an assembly machine at the second stage assembles the m components together to finish the job. Sequence-independent setup times on all machines are considered separated from processing time. Since all factories are identical, processing, assembly, and setup times for job j are the same in each factory, respectively. Let $t_{j,k}$ and $s_{j,k}$ be processing and setup times of job j on machine k at the first stage, p_j and r_j be assembly and setup time of job j at the second stage, respectively. The problem can be divided into two sub-problems, how to assign n jobs to f factories (SP1) and sequence jobs for each factory (SP2). Fig. 1 shows the relationship of the two sub-problems. Our goal is to determine the job assignment to factories and the job sequence in each factory with the minimum TCT.

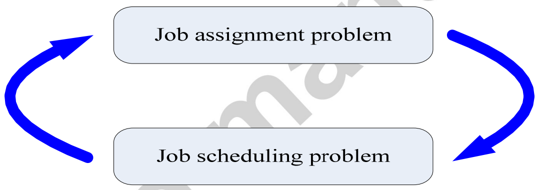

Fig.1. Relationship of the two sub-problems

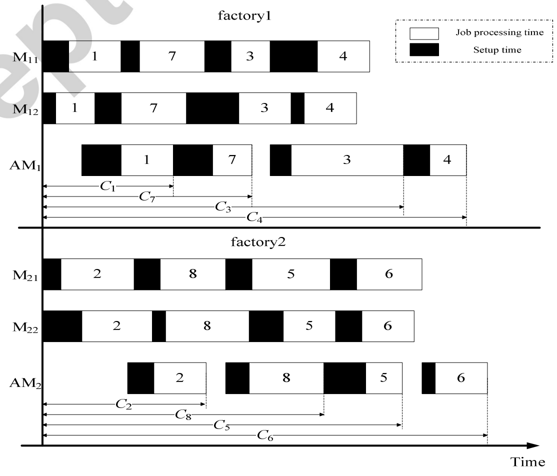

Fig. 2. A schedule for DTSAFSP-TCT with $f = 2$, $m = 2$, and $n = 8$.

TCT of a schedule $\pi = (\pi^1, \pi^2,\ldots, \pi^f)$ can be calculated as follows.

$$C_{\pi_1^i} = \max\left\{ \max_{k=1,2,\ldots m} \left(s_{\pi_1^i,k} + t_{\pi_1^i,k} \right), r_{\pi_1^i} \right\} + p_{\pi_1^i},\ i = 1, 2, \ldots, f, \qquad (1)$$

$$C_{\pi_v^i} = \max\left\{\max_{k=1,2,\ldots m}\left\{\sum_{\rho=1}^{v}\left(s_{\pi_\rho^i,k} + t_{\pi_\rho^i,k}\right)\right\}, C_{\pi_{v-1}^i} + r_{\pi_v^i}\right\} + p_{\pi_v^i},\ i = 1, 2, \ldots, f;\ v = 2, \ldots, n_i, \tag{2}$$

$$TCT = \sum_{i=1}^{f}\sum_{v=1}^{n_i} C_{\pi_v^i}, \tag{3}$$

Fig. 2 illustrates a schedule for a DTSAFSP-TCT with $f = 2$, $m = 2$, and $n = 8$, where jobs 1, 3, 4, and 7 are assigned to factory 1, whereas job 2, 5, 6, and 8 to factory 2. The job sequences in factory 1 and 2 are 1, 7, 3, 4, and 2, 8, 5, 6, respectively.

2.2. A property of DTSAFSP-TCT

In this section, a property of DTSAFSP-TCT is given as follows.

Theorem 1. In each optimal solution of the DTSAFSP-TCT, at least one job is assigned to each factory (if $n > f$).

Proof. Let $x^* = (x^*_{1,1}, x^*_{1,2}, \ldots, x^*_{1,n_1}; x^*_{2,1}, x^*_{2,2}, \ldots, x^*_{2,n_2}; \ldots; x^*_{f,1}, x^*_{f,2}, \ldots, x^*_{f,n_f})$ be an optimal solution for an instance of the DTSAFSP-TCT. Without loss of generality, assume that no jobs are assigned in factory 1 of the optimal solution x^*. Let C_j be the completion time of job j, where $j = 1, 2, \ldots, n$. Since $n > f$, at least one factory is allocated no less than two jobs for x^*. Without loss of generality, let factory 2 be the factory which has no less than two jobs. Move the last job $r = x^*_{2,n_2}$ in the sequence of factory 2 to the last position of the factory 1. Before job r is moved from factory 2 to factory 1, its completion time is denoted by C_r. After job r is moved from factory 2 to factory1, its completion time is denoted by $\hat{C}_r$. As factory 2 has no less than two jobs before moving, we obtain $\hat{C}_r < C_r$. Let $\Phi = \sum_{j=1}^{n} C_j$ be the *TCT* value before job moving operation, whereas $\hat{\Phi}$ be the TCT value after job moving operation. As only the completion of job r has been changed, we have $\hat{\Phi} = \hat{C}_r + \sum_{j=1,\, j\neq r}^{n} C_j < C_r + \sum_{j=1,\, j\neq r}^{n} C_j = \sum_{j=1}^{n} C_j = \Phi$. That is, $\hat{\Phi} < \Phi$. This contradicts with that x^* is an optimal solution. Therefore, the assumption cannot be hold. Theorem 1 is proved. □

Theorem 1 illustrates that it is not an optimal solution if one factory has no job (if $n > f$). Thus, in Section 3, we generate initial solutions in which there is at least one job in each factory.

3. Algorithms for the DTSAFSP-TCT

Owing to the NP-hardness of the DTSAFSP, it is unlikely to obtain optimal solutions to the DTSAFSP-TCT in polynomial time for large size instances. Thus, in this paper, three hybrid meta-heuristics, which aim to find near-optimal solutions with acceptable computational time, have been developed. At first, an extended SPT (ESPT) heuristic, which is used to generate a better initial solution for meta-heuristics, is presented in Subsection 3.1. Subsequently, three hybrid meta-heuristics (HVNS, HGA-RVNS and HDDE-RVNS) are proposed in Subsection 3.2~3.4, respectively.

3.1. An extended SPT heuristics

In this section, we present a heuristic for the DTSAFSP-TCT. The heuristic can be used to obtain a better initial solution for meta-heuristics.

For the DTSAFSP-TCT, if there is only one machine in each factory, the DTSASFSP-TCT will be conducted to the identical parallel machine scheduling problem with the *TCT* criteria (IPMSP-TCT). Pinedo (2002) [25] showed that SPT heuristic is optimal for the IPMSP-TCT. Inspired by this conclusion, combining with the two-stage assembly flowshop environments, we proposed an extended SPT heuristic (ESPT) for the DTSAFSP-TCT. The basic steps of the ESPT are as follows.

ESPT Algorithm:

```
Step 1: Set $C^i_{\max} \leftarrow 0$ and $TP_i \leftarrow 0$, $i = 1, 2,\ldots, f$, where $C^i_{\max}$ is the makespan of factory $i$;
        Computing $\theta_j \leftarrow \max\left\{\max_{k=1,2,..m}(s_{j,k}+t_{j,k}),\ r_j\right\}+p_j$ for $j = 1, 2, \ldots, n$. Sequence jobs on the
        ascending order of $\theta_j$, obtain a job sequence $\pi = (j_1, j_2,\ldots, j_n)$.
Step 2: For $i = 1: f$ do
            Step 2.1: Assign job $j_i$ to factory $i$ and let $\pi^i \leftarrow (j_i)$;
            Step 2.2: Set $C_{j_i} \leftarrow \theta_{j_i}$;
Step 3: For $h = f + 1: n$ do
            Step 3.1: For $i = 1: f$ do
                          add job $j_h$ to the last position in the schedule of factory $i$ and form a new
                          sequence of jobs $\pi^i \cup j_h$; Compute the makespan of $\pi^i \cup j_h$;
            Step 3.2: Find the number $u$, where $u$ is the index of the factory with the minimum
                      makespan when assigned job $j_h$.
            Step 3.3: Assigned job $j_h$ to factory $u$ and $\pi^u \leftarrow \pi^u \cup j_h$; Compute $C^u_{\max}$, where $C^u_{\max}$
                      is the current makespan of the factory $u$ which is assigned job $j_h$. Set
                      $C_{j_h} \leftarrow C^u_{\max}$.
Step 4: Compute $TCT = \sum_{h=1}^{n} C_{j_h}$.
Step 5: Output the solution and its $TCT$ value.
```

3.2. HVNS algorithm

Variable neighborhood search (VNS) is a meta-heuristic optimization method developed by Mladenovic and Hansen [25]. VNS has been successfully applied to several combinatorial optimization problems [29-33]. The main mechanism of VNS algorithm is to enhance the performance of a local search method by changing neighborhood structures. VNS consists of two parts: shaking and local search. The former is to change the neighborhood structure, so that the procedure can jump out of local optimal solution to seek the global optimum. The latter is to search for the local optima in the same neighborhood structure.

The basic steps of VNS are as follows [25].

Step 1: Initialization. Select the set of neighborhood structures N_κ, $\kappa = 1, 2,\ldots, \kappa_{\max}$; Find an initial solution s_0, which is a randomly generated solution. Set a stopping condition.

```
Step 2: While stopping condition is not met do
        Step 2.1: Set $\kappa \leftarrow 1$;
        Step 2.2: While $\kappa \leq \kappa_{\max}$ do
                Step 2.2.1: Shaking. Randomly generate a solution $s'$ from the $\kappa$th
                            neighborhood of $s_0$ ($s' \in N_\kappa(s_0)$);
                Step 2.2.2: Local search. Apply a local search method with $s'$ as initial
                            solution and obtain a solution $s''$;
                Step 2.2.3: Move or not. If $s''$ is better than $s_0$, then $s_0 \leftarrow s''$, $\kappa \leftarrow 1$;
                            otherwise, set $\kappa \leftarrow \kappa + 1$.
        End while
End while
```

In our VNS algorithm, an initial solution is generated by the ESPT heuristic. Four neighborhoods structures are used for shaking and a reduced VNS-based (RVNS) method is employed for local search. The main part of the proposed VNS algorithm is comprised of solution representation and generation, neighborhood structures, and local search. The following subsections will introduce these parts in detail.

3.2.1. Solution representation and generation

In our hybrid VNS algorithm, a solution of the DTSAFSP is encoded as an $n + f - 1$ dimensional vector $s = (s_1, s_2, \ldots, s_j, \ldots, s_{n+f-1})$, where $s_j \in \{1, 2, \ldots, n, *\}$. The solution consists of n indexes of jobs and $f - 1$ separators, where a separator can be seen as a virtual job, denoted as $*$. The $n + f - 1$ dimensional vector is divided into f sections by separators. Such an $n + f - 1$ dimensional vector can be decoded into a schedule as follows.

Section i represents all information of the solution in factory i; that is, all jobs in section i will be assigned to factory i and they are processed in order of the jobs in the section i.

7	5	*	3	1	4	*	8	2	6

Fig. 3. Solution representation for a problem with 8 jobs and 3 factories

For example, Fig. 3 shows a solution representation of the DTSAFSP with 8 jobs and 3 factories. The job sequence is separated into 3 sections by 2 separators. There are 3 sections in the solution representation, (7 5), (3 1 4), and (8 2 6). Jobs 7, 5 are assigned to factory 1, jobs 3, 1, 4 to factory 2 and 8, 2, 6 to factory 3, respectively. The job sequences in factories 1, 2, and 3, are 7 5, 3 1 4, and 8 2 6, respectively. Fig. 4 presents the decoding process.

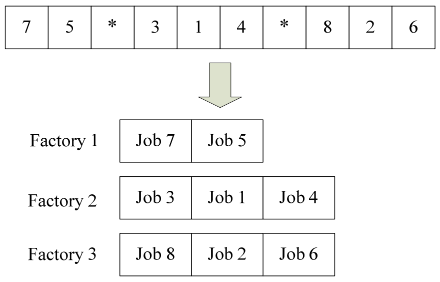

Fig. 4. Process of decoding

In our HVNS algorithm, an initial solution is generated by the ESPT heuristic.

3.2.2. Neighborhood structures

Let $s = (s_1, s_2,\ldots, s_{n+f-1})$ denote a solution of DTSAFSP-TCT. $N_\kappa(s)$ represents the sets of solutions in the κth neighbourhood of s. In our VNS algorithm, four neighborhoods N_κ, $\kappa = 1, 2, 3, 4$ ($\kappa_{\max} = 4$) are used as follows.

1) N_1: Insert neighborhood.

All solutions in $N_1(s)$ are generated by an insert operation from s. A solution $s' \in N_1(s)$ is generated by insert operation as follows. At first, randomly choose two positions u and v, where at most one of them is a separator and $u < v$. Then move the job (the job can be a separator) at position u to position v, whereas all jobs at position k, with $k = u+1,\ldots, v$, are shifted one element forward along the solution.

If $s_u = s_v \neq *$, and there are no separators between positions u and v, the insert operation represents moving a job to another position in the same factory. If $s_u = s_v \neq *$, and there are separators between positions u and v, the insert operation represents moving a job from one factory to another factory. If either s_u or s_v is a separator, the insert operation represents moving a job from one factory to the first position in another factory.

Fig. 5 shows an example of insert operation where $u = 3$ and $v = 7$ are two positions. job 3 at position 3 is moved to position 7, whereas all elements in position k, with $k = 4, 5, 6$, are shifted one position forward.

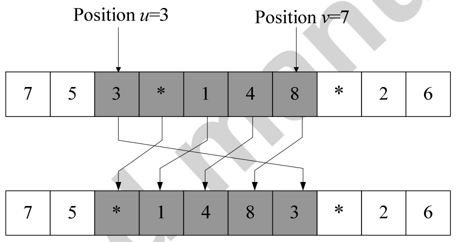

Fig. 5. An example of insert operation

2) N_2: Swap neighborhood.

All solutions in $N_2(s)$ are generated by a swap operation from s. A solution $s' \in N_2(s)$ is generated as follows. Randomly choose two position u and v. The jobs in position u and v cannot be separators simultaneously**.** Then **s**wap s_u and s_v and obtain the solution $s' = (s_1,\ldots, s_{u-1}, s_v, s_{u+1}\ldots, s_{v-1}, s_u, s_{v+1},\ldots, s_{n+f-1})$.

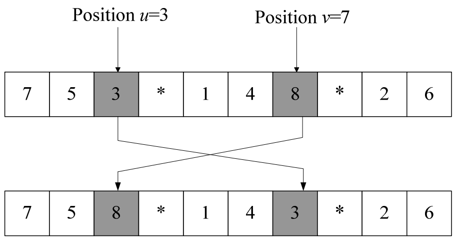

Fig. 6. An example of swap operation

If there are no separators between s_u and s_v, the swap operation is swapping two jobs in the same factory; else, it is swapping a job in one factory with a job in another factory. Fig. 6 illustrates an example of the swap operation, where $u = 3$ and $v = 7$ and jobs in position 3 and position 7 are swapped.

3) N_3: Inverse neighborhood.

All solutions in $N_3(s)$ are generated by inverse operations from s. A solution $s' \in N_3(s)$ is generated as follows. Randomly choose two difference positions u and v, and then inverse the jobs between position u and v.

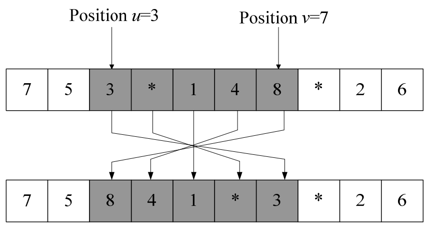

Fig. 7. An example of inverse operation

Fig. 7 illustrates the inverse operation, where positions 3 and 7 are chosen and jobs between positions 3 and 7 are inversed.

4) N_4: λ-Exchange neighborhood.

All solutions in $N_4(s)$ are generated by a λ-Exchange operation from s. A solution $s' \in N_4(s)$ is generated as follows. Randomly choose λ positions in s, and rearrange these λ jobs in λ positions. Usually, $3 \leq \lambda \leq n$.

Fig. 8 shows an example of 4-Exchange operation where 4 positions 3, 5, 6, and 9 are chosen. The jobs in these 4 positions are reassigned, and jobs 3, 1, 4, and 2 are assigned to positions 6, 9, 5, and 3, respectively.

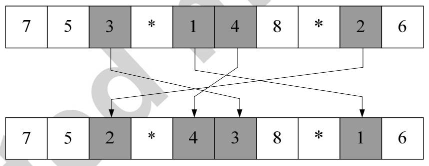

Fig. 8. An example of λ-Exchange operation

3.2.3. Local search

Many local search methods can be used in VNS procedures. In our hybrid VNS algorithm, aiming to perform local search quickly and efficiently, a reduced variable neighborhood search (RVNS) algorithm is selected as the local search step and its procedure is as follows.

```
Step 1: Input initial solution $s_0$ and set maximum iteration number $inloop_{\max}$;
Step 2: Set $inloop \leftarrow 0$ and $flag \leftarrow 0$;
Step 3: While $inloop < inloop_{\max}$ do
        Step 3.1: $inloop \leftarrow inloop+1$;
        Step 3.2: If $flag == 0$ then randomly generate a solution $s \in N_2(s)'$ by insert operation;
                  else randomly generate $s' \in N_2(s)$ by swap operation;
        Step 3.3: If $TCT(s') < TCT(s_0)$, then $flag \leftarrow 0$, $s_0 \leftarrow s'$ ; else $flag \leftarrow (flag + 1) \bmod 2$;
    End While
```

3.3 *HGA-RVNS algorithm*

Genetic algorithm (GA) is a bio-inspired optimization method that is widely used to solve many combinatorial optimization problems [10, 13-16, 34]. At first, genetic algorithms generate initial population of individuals that will be evoluated. When the evolution of individuals is performed, the operations such as selection, crossover and mutation are carried out to generate a new generation of individuals. For the solution to escape from the local optima, the RVNS-based local search procedure is performed in each individual with a local search probability. The main part of the GA-RVNS is comprised of population initialization, selection, crossover, mutation, and RVNS-based local search, etc. The following subsections will introduce these steps in detail.

3.3.1. Chromosome representation and population initialization.

Chromosome representation in our HGA-RVNS algorithm is the same as that of the proposed HVNS algorithm in section 3.2.1. Let PS denote the population size. According to Theorem 1, each factory is allocated at least one job in an optimal solution. There, in order to obtain better initial solutions, we generate them that satisfy this condition. The initial PS individuals can be generated as follows. At first, randomly generate $PS - 1$ permutation sequence of n jobs and for each individual randomly insert $f - 1$ separators to distinct the different jobs in the different factories. Then generate an individual by the ESPT algorithm in Section 3.1.

3.3.2. Evaluation function

In the developed HGA-RVNS algorithm, the fitness value of a chromosome *chm* is defined as follows.

$$fit(chm) = 1/TCT(chm), \tag{4}$$

where $TCT(chm)$ is the objective function value of chromosome *chm*.

3.3.3. Selection

In the proposed HGA algorithm, the simulated roulette method is performed to select the parent chromosome. Here the chromosome's selection probability $p_s(chm_j)$ is directly proportional to its fitness value.

$$p_s(chm_j) = \frac{fit(chm_j)}{\sum_{chm_j=1}^{PS} fit(chm_j)}, \qquad j = 1, 2, \ldots, PS \tag{5}$$

Calculate the cumulative probability $q(chm_k) = \sum_{j=1}^{k} p_s(chm_j)$, $k = 1, 2, \ldots, PS$ and compare it to the random number r with a uniform distribution on [0, 1]. If r is not more than $q(chm_1)$, the first chromosome is selected. Otherwise, the kth chromosome is selected ($2 \le k \le PS$) when $q(chm_{k-1}) < r < q(chm_k)$.

3.3.4. Crossover and mutation

Crossover and mutation are important operators of GA. In crossover operation, offsprings will be generated by exchanging some genes of two parents. Many crossover operators had been presented during the recent years [35]. Murata et al. [35] showed that the two-point crossover is effective for the flow shop scheduling problem. Therefore, the two-point crossover methods are used in this study. An example of the two-point crossover method is illustrated in Fig. 9. As can be seen from Fig. 9, the jobs between two randomly selected positions u = 4 and v = 7 are always inherited from one parent to a child, and the other jobs are placed in the order of their appearance in the other parent.

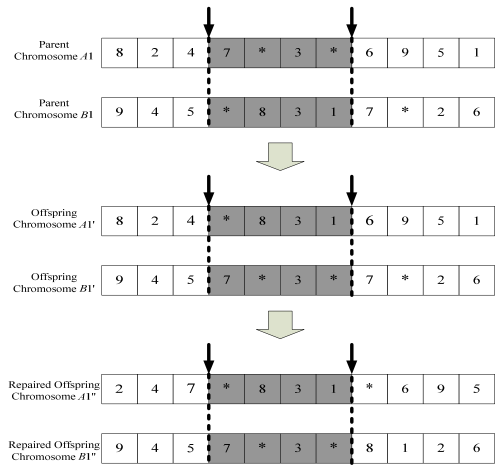

Fig. 9. An example of the two-point crossover

In order to prevent premature and fall into local optimum, mutation operation is essential in a GA. In this paper, we randomly choose one mutation operation from insert, swap, and inverse moves the same as those in Section 3.2.2, and then perform it on each individual with a mutation probability P_m, where $0 \leq P_m \leq 1$.

3.3.5. Flowchart of hybrid GA-RVNS algorithm

Fig. 10 illustrates the Flowchart of the proposed HGA-RVNS algorithm. In order to achieve a well balance between local search and global search, in the algorithm, the same RVNS method as that in HVNS is performed on each chromosome of the current generation with a probability *PM*.

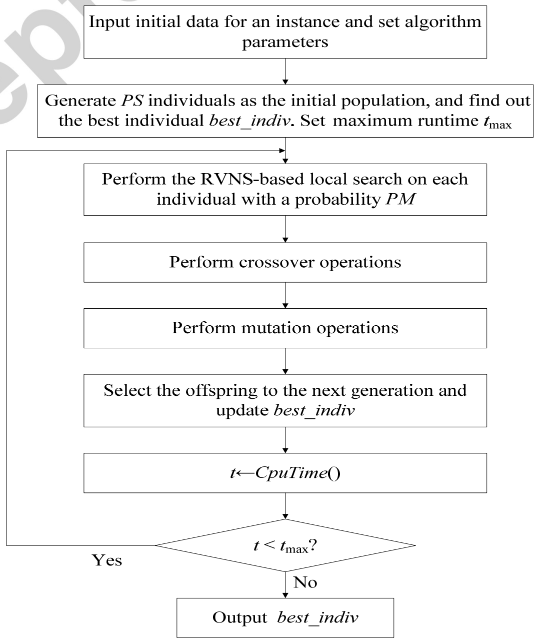
Input initial data for an instance and set algorithm parameters
Generate PS individuals as the initial population, and find out the best individual best_indiv. Set maximum runtime t_{max}
Perform the RVNS-based local search on each individual with a probability PM
Perform crossover operations
Perform mutation operations
Select the offspring to the next generation and update best_indiv
t←CpuTime()
$t < t_{max}$?
Yes
No
Output best_indiv

Fig. 10. Flowchart of the proposed HGA-RVNS algorithm.

3.4 HDDE-RVNS algorithm

3.4.1 Basic differential evolution (DE) algorithm

DE is a stochastic population-based heuritic proposed by Storn & Price (1995) [36] for global optimization over continuous search spaces. In a DE, a floating-point encoding method is adopted for each individual. Firstly, it randomly generates a population of *PS* target individuals. Secondly, *PS* new candidate solutions are created by mutation and crossover operators. Finally, selection operator is performed to select the new target individuals to the next generation from the previous. Let $X_{i,g} = [x_{i,1,g}, x_{i,2,g},\ldots, x_{i,N,g}]$, $V_{i,g} = [v_{i,1,g}, v_{i,2,g},\ldots, v_{i,N,g}]$, and $U_{i,g} = [u_{i,1,g}, u_{i,2,g},\ldots, u_{i,N,g}]$ represent the *i*th target, mutant, and trial individuals at iteration *g*, respectively. They are all *N* dimension vectors. The procedure of the basic DE algorithm is as follows [24, 36].

Basic DE Algorithm:

Step 1: Initialization. Set the size of the population (*PS*), scale factor (*F*), crossover probability (*CR*), and the maximum iterations *g*_max. Randomly generate *PS* individuals as the initial population. Let *best_indiv* be the best individual found so far, and $g \leftarrow 0$.

Step 2: While $g < g$_max do

Step 2.1: Mutation operation. Generate mutant individuals $V_{i,g}$, $i = 1,2,\ldots, PS$ as follows:

$$V_{i,g} \leftarrow X_{r_1,g} + F \times (X_{r_2,g} - X_{r_3,g}), \tag{6}$$

where r_1, r_2 and r_3 are three different integers which are randomly chosen from [1, *PS*], and each of them is different from the integer *i*. $F > 0$ is a scale factor.

Step 2.2: Crossover operation. Generate trial individuals $U_{i,g+1} = [u_{i,1,g+1}, u_{i,2,g+1}, \ldots, u_{i,N,g+1}]$, $i = 1, 2,\ldots, PS$, as follows:

$$u_{i,j,g+1} \leftarrow \begin{cases} v_{i,j,g+1}, \ if\ (rand() \le CR)\ or\ j = \eta_j \\ x_{i,j,g}, \ otherwise. \end{cases}, \tag{7}$$

where $CR \in (0,1)$ is a crossover probability, *rand*() indicates a uniform random number between 0 and 1, and η_j represents an integer randomly chosen from the set $\{1, 2,\ldots, N\}$.

Step 2.3: Selection operation. For a minimization problem, each new target individual can be generated as follows:

$$X_{i,g+1} \leftarrow \begin{cases} U_{i,g+1} & if\ \ f(U_{i,g+1}) < f(X_{i,g+1}) \\ X_{i,g} & otherwise, \end{cases}, \tag{8}$$

where $f(\cdot)$ is the objective function.

Step 2.4: update *best_indiv* and *f*(*best_indiv*) and set $g \leftarrow g+1$.

End While

Step 3: Output *best_indiv* and *f*(*best_indiv*).

3.4.2. Hybrid Discrete differential evolution and reduced VNS (HDDE-RVNS) algorithm

Discrete job sequence and job assignment cannot be generated by the basic DE algorithm which is originally designed for continuous optimization problems. Thus, we will propose a novel hybrid discrete differential evolution algorithm combined with reduced VNS-based local search (HDDE-RVNS) for DTSAFSP-TCT. Solution representation and initialization, mutation, and crossover operators will be presented as follows.

1) Solution representation and initialization

In this section, we propose a job-permutation based solution encoding method. It can be seen from Theory 1 that each factory is allocated at least one job in an optimal solution. Therefore, aimed at obtaining better initial solutions for HDDE-RVNS, we generate an initial solution that satisfies this condition. A solution of the DTSAFSP is encoded as an $n+f-1$ dimensional vector $x = (x_1, x_2,\dots, x_j,\dots, x_{n+f-1})$, where $x_j \in \{1, 2,\dots, n, n+1,\dots, n+f-1\}$. In x, if $x_j \le n$, it is a job index; else, it can be seen as a separator. Thus, there are $f-1$ separators in x and x is divided into f sections by separators. Fig. 11 shows an example of solution decoding of our HDDE-RVNS (3 factories and 8 jobs).

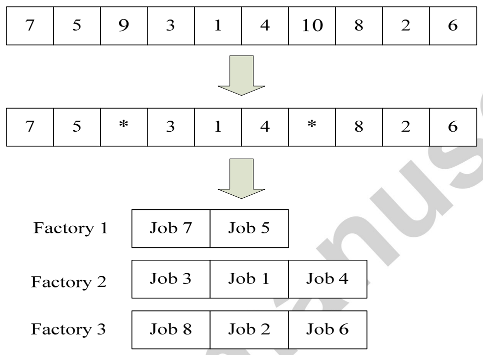

Fig. 11. An example of decoding process of the proposed HDDE-RVNS

The procedure of population generalization is as follows. Generate one solution by ESPT and randomly generate the remaining ($PS-1$) permutation-based individuals with $n+f-1$ dimension.

2) Mutation operator

Since the mutation operator of basic DE cannot be applied directly for discrete optimization problems, it must be redesigned to handle the discrete vector. In our HDDE-RVNS, mutation operator is adopted as follows.

$$V_i = X_{r_1} \oplus F \otimes (X_{r_2} \ominus X_{r_3}), \tag{9}$$

where $\ominus$, $\otimes$ and $\oplus$ denote three operators, namely subtraction, multiplication, and addition operators, respectively, and they are defined as follows.

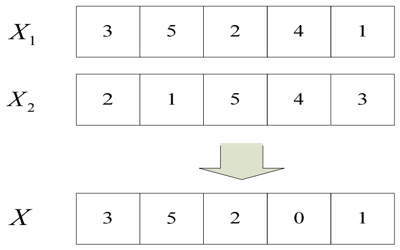

Fig. 12. An example of the subtraction operator

Definition 1. Subtraction operator ⊖

Let $X_1 = (x_{1,1}, x_{1,2}, \ldots, x_{1,n+f-1})$ and $X_2 = (x_{2,1}, x_{2,2}, \ldots, x_{2,n+f-1})$ be two $(n + f - 1)$ dimension vectors where $x_{1,j}, x_{2,j} \in \{1, 2,\ldots, n + f - 1\}$. Define $X_1 \ominus X_2$ to be a $(n + f - 1)$ dimension vector $X = (x_1, x_2, \ldots, x_{n+f-1})$, where $x_j = 0$ if $x_{1,j} = x_{2,j}$; $x_j = x_{1,j}$, otherwise.

Fig. 12 shows a simple example of the subtraction operator between X_1 and X_2, where $n = 4, f = 2$, $X_1 = (3, 5, 2, 4, 1)$ and $X_2 = (2, 1, 5, 4, 3)$.

Definition 2. Multiplication operator ⊗.

Let $X = (x_1, x_2, \ldots, x_{n+f-1})$ be a $(n + f - 1)$ dimension vector where $x_j \in \{0, 1, 2,\ldots, n + f - 1\}$, a be a real number between (0, 1) and $r = (r_1, r_2,\ldots, r_{n+f-1})$ be a $(n + f - 1)$ dimension vector where r_j is a uniform random number generated between [0, 1]. Define $a \otimes X$ to be a $(n + f - 1)$ dimension vector $\Delta = (\delta_1, \delta_2, \ldots, \delta_{n+f-1})$ where $\delta_j = x_j$ if $a > r_j$; $\delta_j = 0$, otherwise.

Fig. 13 shows an example of the multiplication operator ⊗ between a and X, where $n = 4, f = 2$, $a = 0.5$ and $X = (3, 5, 2, 0, 1)$.

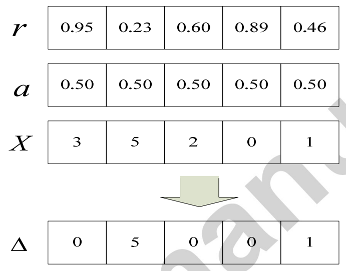

Fig. 13. An example of the multiplication operator

Definition 3. Addition operator ⊕

Let $X_1 = (x_{1,1}, x_{1,2}, \ldots, x_{1,n+f-1})$ and $\Delta = (\delta_1, \delta_2, \ldots, \delta_{n+f-1})$ be two $(n + f - 1)$ dimension vectors where $x_{1,j} \in \{1, 2,\ldots, n + f - 1\}$ and $\delta_j \in \{0, 1, 2,\ldots, n + f - 1\}$, $j = 1, 2,\ldots, n + f - 1$. Define $X_1 \oplus \Delta$ to be a $(n + f - 1)$ dimension vector $V = (v_1, v_2,\ldots, v_{n+f-1})$, where $v_j \in \{1, 2,\ldots, n + f - 1\}$. V is computed as follows. At first, set $v_j = x_{1,j}$, $j = 1,2,\ldots, n + f - 1$. Then compare v_j and δ_j for each $j = 1$ to $n + f - 1$. If $\delta_j \neq 0$, find the position j' in which $v_{j'} = \delta_j$, and then swap v_j and $v_{j'}$; otherwise, v_j remains unchanged.

Fig. 14 shows an example of the addition operator ⊕ between X_1 and Δ, where $n = 4, f = 2$, $X_1 = (2, 4, 1, 5, 3)$ and $\Delta = (0, 5, 0, 0, 1)$.

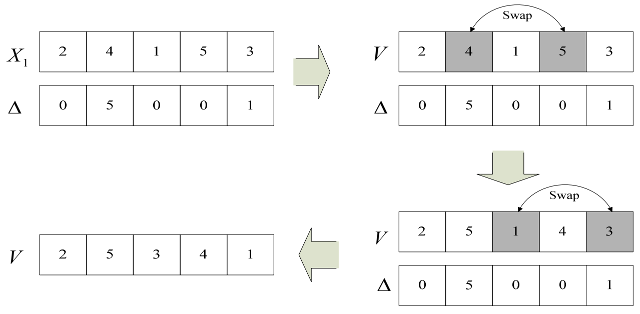

Fig. 14. An example of the addition operator

Thus, with Definitions 1, 2 and 3, the mutation operator can be performed on the proposed individual represented by the discrete-value vector.

3) Crossover operator

In crossover operation, a trial individual U_i can be generated by combining a target individual X_i and a mutant individual V_i. We adopt the similar crossover operator as that of in Section 3.3.4. The distinct is in that the crossover points in our HDDE-RVNS are selected with a probability *CR*. Fig. 15 is an example of the crossover operation for HDDE-RVNS, where *CR* is set to 0.5. As can be seen from Fig. 15, the jobs in position 2, 4, 7, 8 and 10 are randomly selected with the crossover probability *CR* are always inherited from one target individual V_i to a trial individual, and the other jobs are placed in the order of their appearance in the mutant individual X_i.

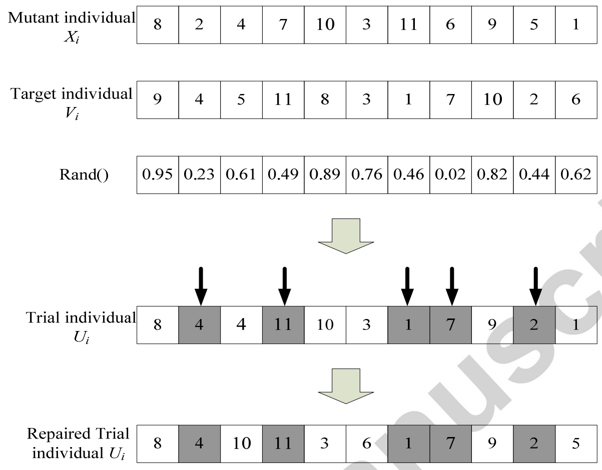

Fig. 15. An example of the crossover operation in the HDDE-RVNS

4) Flowchart of the proposed HDDE-RVNS algorithm

Based on the above design, the flowchart of the proposed HDDE-RVNS for the DTSAFSP-TCT can be seen in Fig. 16. In HDDE-RVNS, we use the same selection operator as that in the basic DE algorithm and perform the same RVNS-based local search as that in HGA-RVNS.

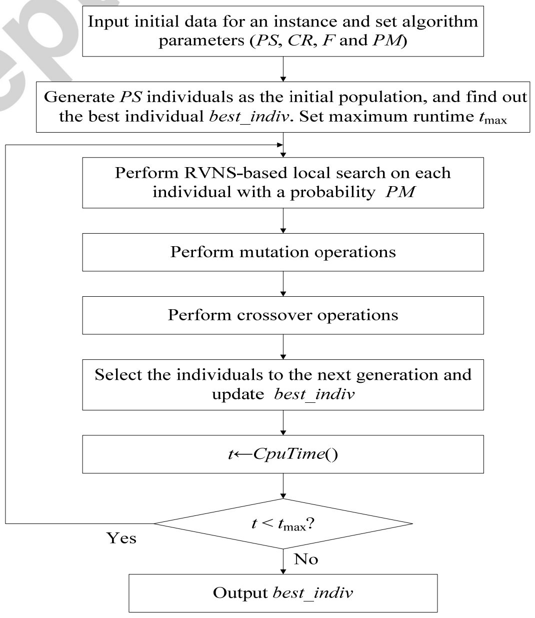
Input initial data for an instance and set algorithm parameters (*PS*, *CR*, *F* and *PM*)
Generate *PS* individuals as the initial population, and find out the best individual *best_indiv*. Set maximum runtime $t_{\max}$
Perform RVNS-based local search on each individual with a probability *PM*
Perform mutation operations
Perform crossover operations
Select the individuals to the next generation and update *best_indiv*
$t \leftarrow CpuTime()$
$t < t_{\max}$?
Yes
No
Output *best_indiv*

Fig. 16. Flowchart of the proposed HDDE-RVNS algorithm

4. Computational experiments

4.1. Experimental design

We implement HVNS, HGA-RVNS, and HDDE-RVNS in Matlab 7.0. All experiments were run on a PC with Intel Core i7-2600 CPU, Windows 7, 3.4 GHz and 4GB RAM. To evaluate performances of the proposed algorithms, computational experiments were performed on 80 randomly generated problems with the number of jobs 20, 50, 100, 200, or 500, the number of machines at stage one 2, 4, 6, or 8, and the number of factories 2, 3, 4, or 6. Both the processing times of each job at stage one and two were generated from Unidrnd(1,100), where Unidrnd(1,100) is the discrete uniform distribution with range [1, 100]. Setup times on both stages were drawn from Unidrnd(1, 20). The problem data were summarized in Table 1. Aimed at allowing more computation time as the number of jobs increases, the stopping criteria of meta-heuristics for many scheduling problems are set to a CPU time which is dependent to number of jobs (Ruiz and Alcaraz 2006, Naderi *et al*. 2009, Pan and Ruiz 2012). Therefore, in this study, the stop criterion used to test all the algorithms for the same instance is set to a CPU time limit fixed to 500×*n* milliseconds.

Table 1
Data for generating test problems

Data	Value
Number of jobs (n)	20, 50, 100, 200, 500
Number of machines at stage one (m)	2, 4, 6, 8
Number of factories (f)	2, 3, 4, 6
Processing time at the first stage ($t_{j,k}$)	Unidrnd(1, 100)
Processing time at the second stage (p_j)	Unidrnd(1, 100)
Setup time at stage one ($s_{j,k}$)	Unidrnd (1, 20)
Setup time at stage two (r_j)	Unidrnd (1, 20)

4.2. Parameter tuning

It is extremely importance to tune the parameters since the performance of a meta-heuristic is usually sensitive to them. In this paper, the proposed HGA-RVNS has five parameters: P_c, P_m, PS, PM and $Inloop_{\max}$, and HDDE-RVNS also has five parameters: CR, F, PS, PM and $Inloop_{\max}$. The proposed VNS only has one parameter: $Inloop_{\max}$. Taguchi method is an effective design of experiment approach [37] and has been widely used to parameter tuning of meta-heuristics [38-41]. Thus, in this paper, we applied Taguchi method to tune the parameters of HGA-RVNS and HDDE-RVNS. The considered levels for HGA-RVNS and HDDE-RVNS are listed in Tables 2 and 3, respectively.

According to the numbers of parameters and the number of factor levels, $L_{16}(4^5)$ is chosen as the fittest orthogonal array design that fulfils the minimum requirements. The selected Taguchi design has 16 different combinations of parameter levels. 8 instances (four small and four large instances) from the combinations of (n = {20, 50, 100, 200, 500}, m = {2, 4, 6, 8}, and f = {2, 4, 6}) are randomly generated for each trial. Aimed at obtaining more reliable results, each instance is run 20 times.

The performance measure of the proposed algorithms was calculated as the percentage of the relative percentage deviation (RPD) from the obtained solution to the best one. The formula of RPD is given as follows:

$$RPD = \frac{TCT - TCT^*}{TCT^*} \times 100 \, , \tag{10}$$

where TCT is the objective value obtained for a given algorithm and instance and TCT^* is the best solution provided so far by all the three proposed algorithms. Furthermore, the average relative percentage deviation ($ARPD$) is calculated as the performance statistics.

Table 2. The orthogonal array $L_{16}(4^5)$ of HGA-RVNS and ARPD

Parameters	Factor level			
	1	2	3	4
PS	30	40	50	60
P_c	0.6	0.7	0.8	0.9
P_m	0.10	0.15	0.20	0.25
PM	0.10	0.15	0.20	0.25
$Inloop_{\max}$	50	75	100	125

Table 3. The orthogonal array $L_{16}(4^5)$ of HDDE-RVNS and ARPD

Parameters	Factor level			
	1	2	3	4
PS	20	30	40	50
F	0.6	0.7	0.8	0.9
CR	0.10	0.15	0.2	0.25
PM	0.10	0.15	0.20	0.25
$Inloop_{\max}$	50	75	100	125

Table 4. The orthogonal array $L_{16}(4^5)$ of HGA-RVNS and ARPD (average relative error) for HGA-RVNS

Trial	*Factors*					*ARPD*
	PS	P_c	P_m	PM	$Inloop_{\max}$	
1	1	1	1	1	1	0.730
2	1	2	2	2	2	1.014
3	1	3	3	3	3	0.975
4	1	4	4	4	4	1.211
5	2	1	2	3	4	1.179
6	2	2	1	4	3	1.277
7	2	3	4	1	2	0.978
8	2	4	3	2	1	1.300
9	3	1	3	4	2	1.522
10	3	2	4	3	1	1.399
11	3	3	1	2	4	1.226
12	3	4	2	1	3	1.240
13	4	1	4	2	3	1.378
14	4	2	3	1	4	1.240
15	4	3	2	4	1	1.617
16	4	4	1	3	2	1.446

Table 5. ARE and rank of each parameter for HGA-RVNS

Level	*PS*	P_c	P_m	*PM*	$Inloop_{\max}$
1	0.983	1.227	1.170	1.047	1.262
2	1.184	1.233	1.263	1.230	1.240
3	1.347	1.199	1.259	1.250	1.218
4	1.420	1.299	1.242	1.407	1.214
Delta	0.438	0.100	0.093	0.360	0.048
Rank	1	3	4	2	5

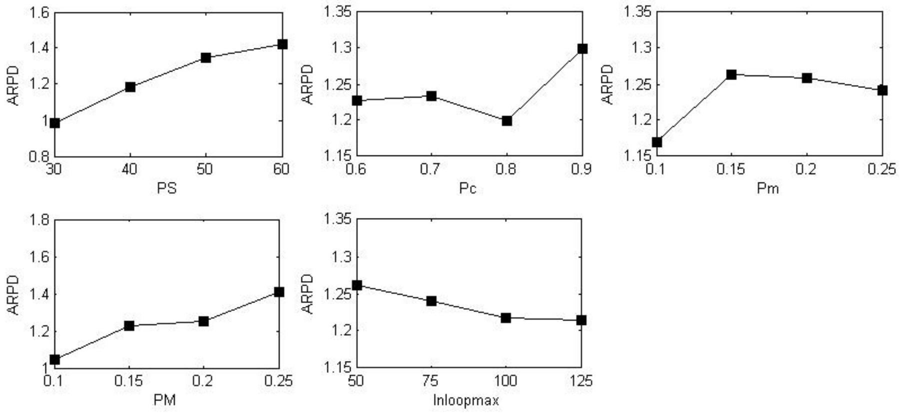

Fig. 17. ARPD plot for each level of the factor of HGA-RVNS

Table 6. The orthogonal array $L_{16}(4^5)$ of HDDE-RVNS and ARPD (average relative error)

Trial	*Factors*					*ARPD*
	PS	*F*	*CR*	*PM*	$Inloop_{\max}$	
1	1	1	1	1	1	0.535
2	1	2	2	2	2	0.635
3	1	3	3	3	3	0.657
4	1	4	4	4	4	0.557
5	2	1	2	3	4	0.876
6	2	2	1	4	3	0.841
7	2	3	4	1	2	1.200
8	2	4	3	2	1	1.201
9	3	1	3	4	2	1.179
10	3	2	4	3	1	1.327
11	3	3	1	2	4	1.228
12	3	4	2	1	3	1.239
13	4	1	4	2	3	1.365
14	4	2	3	1	4	1.528
15	4	3	2	4	1	1.449
16	4	4	1	3	2	1.377

For HGA-RVNS, The orthogonal array $L_{16}(4^5)$ and the trend of each factor are shown in Table 4 and Fig. 17, respectively. Table 5 illustrates that the population size *PS* is the most significant parameter. The significant rank of *PM* is 2nd; the significant rank of P_c is 3rd; the significant rank of P_m is 4rd; the significant rank of $Inloop_{\max}$ is 5rd. According to Figure12, a good combination of parameters values for HGA-RVNS are determined as *PS* = 30, *PM* = 0.1, $P_c = 0.8$, $P_m = 0.1$, and $Inloop_{\max} = 125$.

For HDDE-RVNS, Table 6 and Fig. 18 show the orthogonal array $L_{16}(4^5)$ and the trend of each factor, respectively. It can be seen from Table 7 that the population size *PS* is the most significant parameter. *CR* ranks the second, *F* the third, *PM* the fourth and $Inloop_{\max}$ the last. According to Fig.18, a good choice of the parameters for HDDE-RVNS can be obtained as *PS* = 20, *F* = 0.6, *CR* = 0.1, *PM* = 0.25, and $Inloop_{\max} = 100$.

Table 7. ARE and rank of each parameter for HDDE-RVNS

Level	*PS*	*F*	*CR*	*PM*	$Inloop_{\max}$
1	0.596	0.989	0.995	1.126	1.128
2	1.030	1.083	1.050	1.148	1.098
3	1.243	1.134	1.141	1.059	1.026
4	1.430	1.094	1.112	1.007	1.047
Delta	0.834	0145	0.146	0.141	0.102
Rank	1	3	2	4	5

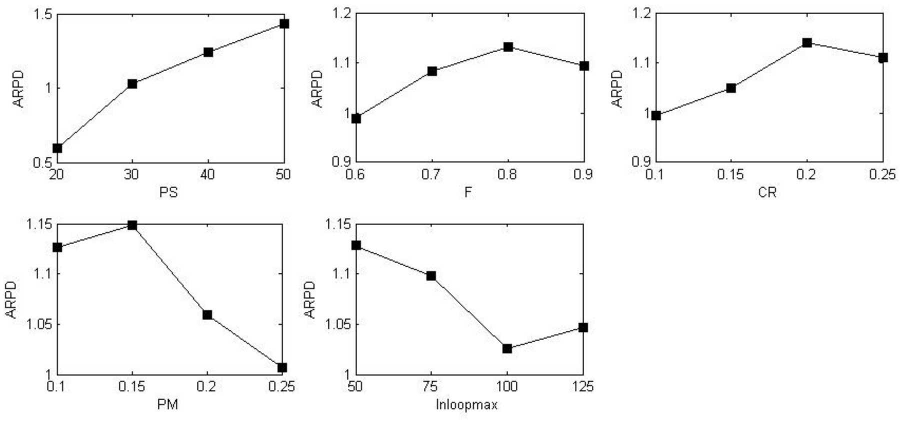

Fig. 18. ARPD plot for each level of the factor of HDDE-RVNS

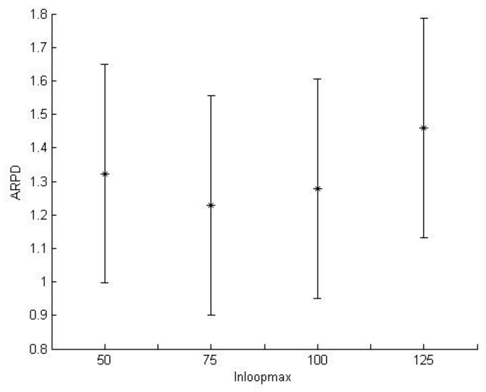

Fig.19. The mean plot and LSD intervals for different levels of $Inloop_{\max}$

For the proposed HVNS, there is only one parameter ($Inloop_{\max}$) to be tuned. The considered level is: 50, 75,100 and 125. Thus, a single factor analysis of variance (ANOVA) technique [37] has been used to analyze the results. Fig. 19 shows the means plot and LSD intervals for different levels of the parameter $Inloop_{\max}$ of HVNS. It can be seen from the figure that $Inloop_{\max}$ of 75 is slightly outperform the other levels although they are statistically similar. Thus, $Inloop_{\max}$ of 75 is chosen for the proposed VNS.

To validate the effectiveness of the local search in HGA_RVNS and HDDE_RVNS, HGA and HDDE without local search step (HGA-NOV and HDDE-NOV) were also coded in Matlab 7.0 and run on the same PC. The parameters of these two algorithms are also tuned by Taguchi methods and their parameters are chosen as follows.

HGA-NOV: $PS = 30$, $P_c = 0.6$, $P_m = 0.1$;

HDDE-NOV: $PS = 50$, $F = 0.6$, $CR = 0.2$.

4.3. Comparison results

In this study, we focus on the distributed two stage assembly flowshop scheduling problem (DTSAFSP). This problem involves two inter-dependent decision problems: (1) How to allocate jobs among factories, and (2) how to schedule the assigned jobs at each factory. The DTSAFSP is more complex than the classical two stage assembly flowshop problem. Its objective is to minimize the total completion time. To the best of our knowledge, there is no research on the distributed two stage assembly flowshop problem with minimum TCT criteria (DTSAFSP-TCT). Therefore, it is a novel distributed scheduling problem (DSP) and there are no algorithms for dealing with this problem in literature. There is also no benchmark for the DTSAFSP-TCT, hence it is hard to compare the results with a greater range of solution methods. Based on three classical meta-heuristics (GA, DE and VNS), we proposed three hybrid meta-heuristics for this novel DSP. So, in the computational results, we consider two main purposes as follows:

1) The effectiveness of three proposed algorithms method (HGA-RVNS, HDDE-RVNS and VNS) against each other.

2) Demonstrate the effectiveness of HGA-RVNS and HDDE-RVNS against simple HGA and HDDE.

At first, we compared the performances of three proposed meta-heuristics against each other. Tables 2~5 summarize the computational results of 80 instances for different combinations of f, n and m. Those results are obtained by running the proposed algorithms 20 times for each instance. The minimum TCT value ($MinTCT$), the average TCT value ($AvgTCT$), the maximum TCT value ($MaxTCT$) and the standard deviation (Std) of 20 replications are listed on the Tables 2~5. Best $MinTCT$ results for each $f \times n \times m$ are in bold type. The results reported in Tables 2~5 show that HVNS, HDDE-RVNS and HGA-RVNS differ in performance with different size.

For the small size instances with $n = 20$ and $n = 50$, it can be seen from Tables 2~5 that HDDE-RVNS yields better $MinTCT$, $AvgTCT$, $MaxTCT$ values with smaller Std values than HVNS and HGA-RVNS with the same computational time limit. Therefore, it is concluded that for small size instances, HDDE-RVNS are more effectively, efficiently and robustly than the other proposed algorithms. For the large size instances with n = 100, 200 and 500, HGA_RVNS yields better $MinTCT$, $AvgTCT$, and $MaxTCT$ values than HVNS and HDDE-RVNS. As we can see, HGA-RVNS apparently outperform the other proposed algorithms for the large size instances.

Table 2
Computational results (HVNS, HDDE-RVNS and HGA-RVNS, *f* = 2).

f	*n*	*m*	HVNS				HDDE-RVNS				HGA-RVNS			
			MinTCT	MaxTCT	AvgTCT	Std	MinTCT	MaxTCT	AvgTCT	Std	MinTCT	MaxTCT	AvgTCT	Std
	20	2	8484	8537	8511	14.8	**8452**	8468	8457	6.9	8454	8564	8508	41.8
	20	4	7032	7126	7083	31.5	**7010**	7045	7030	12.7	7010	7124	7048	34.8
	20	6	8685	8929	8748	70.1	**8641**	8734	8680	26.6	8671	8782	8721	34.6
	20	8	8362	8543	8414	56.3	**8310**	8379	8351	22.3	8373	8535	8444	51.7
	50	2	34316	34817	34557	185.9	**33973**	34184	34112	66.1	34038	34373	34215	113.3
	50	4	41074	42437	41643	465.9	**40793**	40931	40857	45.0	40823	41307	41017	130.2
	50	6	43434	44476	44116	326.1	**43383**	43632	43484	72.0	43476	43959	43639	161.7
	50	8	44470	45320	44772	280.5	**44177**	44400	44317	63.5	44254	44996	44530	202.9
	100	2	132427	134449	133285	650	132155	133007	132568	288.2	**131852**	132671	132142	284
2	100	4	149811	154383	151113	1371.3	148957	149486	149217	184.2	**147842**	149640	148726	600.7
	100	6	161473	164511	162739	983.9	160775	161477	161021	231.5	**159343**	162226	160583	877.9
	100	8	159002	161979	160654	946.1	159044	159603	159409	190	**158427**	159560	158957	367.1
	200	2	523608	525647	524716	742.9	520254	523765	522093	1377.2	**515994**	518339	517054	943.8
	200	4	583826	587420	585510	1501.1	583217	587214	584886	1200.4	**576027**	584293	578181	2596.4
	200	6	601399	609072	604554	2349.8	602333	604658	603534	914.5	**597988**	602392	600562	1802.6
	200	8	618278	627009	623488	3430.3	618850	622741	620550	1546.5	**614237**	622317	617772	3730.6
	500	2	3328474	3351883	3338903	9532.9	3327237	3345594	3335810	7849.3	**3287778**	3299427	3296925	5106.2
	500	4	3550478	3581093	3566453	12425	3580537	3606143	3591829	12271	**3521414**	3530636	3525795	3325.3
	500	6	3673334	3734477	3715445	24916	3721402	3758760	3741739	14217	**3659873**	3703474	3681076	21420
	500	8	3694243	3707857	3701316	59873	3707572	3767081	3740859	21373	**3659289**	3677749	3668341	7026.8

Table 3
Computational results (HVNS, HDDE-RVNS and HGA-RVNS, *f* = 3).

f	*n*	*m*	HVNS				HDDE-RVNS				HGA-RVNS			
			MinTCT	MaxTCT	AvgTCT	Std	MinTCT	MaxTCT	AvgTCT	Std	MinTCT	MaxTCT	AvgTCT	Std
	20	2	6262	6409	6358	41.5	**6256**	6284	6270	7.1	6275	6411	6342	45.9
	20	4	5333	5439	5390	37.2	**5292**	5330	5310	11.9	5309	5416	5357	31.6
	20	6	6565	6748	6637	96.5	**6540**	6631	6594	25.8	6551	6720	6622	55.9
	20	8	6330	6613	6436.1	84.4	**6322**	6384	6351	22.0	6332	6418	6387	27.6
	50	2	24297	24711	24495	126.5	**24189**	24406	24325	70.9	24208	24481	24351	100.3
	50	4	29185	30516	29785	439.4	**29014**	29292	29190	87.7	29167	29708	29327	159.1
	50	6	31035	31638	31390	222.6	**30833**	31077	30950	73.2	31071	31387	31199	90.3
	50	8	31823	32539	32167	239	**31483**	31677	31612	59.5	31643	32276	31862	181.6
	100	2	91366	92642	91913	396.4	90761	91677	91320	314.8	**90323**	91249	90749	275.9
3	100	4	104848	109549	106717	1701.9	103386	104641	104171	407.5	**103049**	104121	103520	375.1
	100	6	112192	115354	113398	940.2	112024	113018	112450	344	**110655**	112150	111505	518.5
	100	8	111448	114837	112663	1228.9	110691	111834	111440	320.3	**110047**	112407	110977	746.7
	200	2	356846	359677	357730	1158.2	355021	356321	355694	520.1	**350457**	353442	352483	1218
	200	4	400420	410773	406555	4158.1	401888	404344	402771	964.7	**394582**	397033	396273	974.1
	200	6	416130	421494	418635	2469.1	413481	418732	415889	1923.8	**408893**	411449	410474	1089.5
	200	8	425895	432032	429918	2492.1	426657	428522	427652	672.8	**422539**	427007	424023	1805.7
	500	2	2251131	2276307	2262864	9121.1	2260413	2273662	2268805	5001.2	**2232708**	2242715	2236864	3939.7
	500	4	2411706	2434183	2423822	8529.1	2432849	2455999	2445306	8250.4	**2370653**	2397768	2387098	10505
	500	6	2510929	2528430	2523866	7277.7	2517250	2551632	2540252	14006	**2485109**	2495127	2490678	3941
	500	8	2522587	2533960	2527223	4498.3	2541806	2564788	2550229	9121.2	**2472428**	2499134	2485523	9559.4

Table 4
Computational results (HVNS, HDDE-RVNS and HGA-RVNS, *f* = 4).

f	*n*	*m*	HVNS				HDDE-RVNS				HGA-RVNS			
			MinTCT	MaxTCT	AvgTCT	Std	MinTCT	MaxTCT	AvgTCT	Std	MinTCT	MaxTCT	AvgTCT	Std
4	20	2	5163	5265	5212	38.4	**5150**	5166	5159	7.1	5163	5282	5212	44.9
	20	4	4454	4569	4514	40.9	**4433**	4457	4446	9.1	4435	4545	4489	31.3
	20	6	5566	5685	5602	33.9	**5497**	5585	5544	23.1	5556	5646	5605	28.2
	20	8	5348	5402	5461	35.4	**5303**	5371	5347	22.1	5342	5453	5387	35
	50	2	19494	19802	19654	94.8	**19299**	19425	19355	38.3	19343	19516	19427	56.1
	50	4	23428	24203	23839	230	**23041**	23268	23189	76.2	23130	23602	23378	142.5
	50	6	24882	25192	24990	104.8	**24424**	24675	24596	82.2	24510	24903	24712	129.2
	50	8	25436	25950	25679	170.7	**25104**	25284	25229	57.4	25182	25568	25367	123
	100	2	71419	72390	71792	267.6	70981	71633	71254	216.1	**70347**	70845	70534	163.2
	100	4	80851	86053	82955	1438.4	80599	81547	81187	309.3	**79933**	80854	80428	307.9
	100	6	87419	88980	88201	418.6	87029	87961	87546	270.4	**86697**	87532	87153	224.4
	100	8	86891	90340	88633	1000.5	86771	87283	87034	178.1	**85387**	87333	86183	574.1
	200	2	272141	274326	273382	871.1	270181	272755	271378	953.4	**268367**	269841	269190	615
	200	4	311066	326176	316234	6187	307855	311246	309946	14318	**301485**	305286	303087	1414.6
	200	6	321322	330919	323430	4187.3	319680	321203	320242	641.8	**313469**	317516	315972	1524.8
	200	8	330639	333190	331853	1181.7	329314	330787	329980	586.7	**322592**	324250	323496	691.7
	500	2	1705613	1719531	1715265	5522.6	1708713	1721034	1715347	6106.6	**1688839**	1697285	1692676	3033.8
	500	4	1844266	1852937	1850060	3547.8	1856815	1875108	1861736	7643.5	**1811078**	1821421	1816130	4158
	500	6	1917737	1947236	1929987	13679	1920915	1945981	1934256	9228.9	**1881212**	1894753	1885362	5508
	500	8	1912951	1935538	1922666	8156.3	1936557	1950494	1943640	5522.4	**1885962**	1895074	1890024	3886.5

Table 5
Computational results (HVNS, HDDE-RVNS and HGA-RVNS, *f* = 6).

f	*n*	*m*	HVNS				HDDE-RVNS				HGA-RVNS			
			MinTCT	MaxTCT	AvgTCT	Std	MinTCT	MaxTCT	AvgTCT	Std	MinTCT	MaxTCT	AvgTCT	Std
6	20	2	4098	4117	4108	8.9	**4097**	4102	4099	1.3	4098	4163	4133	18.6
	20	4	3587	3650	3608.2	20	**3578**	3608	3590	10.1	3611	3666	3638.1	17.4
	20	6	4514	4547.8	4609	30.7	**4514**	4534	4525	6.8	4524	4599	4550	22.9
	20	8	4317	4391	4356.4	26.8	**4317**	4368	4332	15.2	4329	4446	4378	34.5
	50	2	14427	14759	14547	106.2	**14272**	14440	14381	49.2	14299	14538	14390	79.9
	50	4	17291	17880	17625	218.3	**17206**	17361	17318	45	17213	17634	17368	127.4
	50	6	18451	18871	18591	133.1	**18188**	18317	18256	40.9	18241	18584	18354	97.2
	50	8	18737	19130	18972	117.1	**18720**	18932	18819	62.5	18732	19013	18864	96.1
	100	2	50779	51233	50990	154.7	50091	50852	50431	227.8	**49507**	50156	49846	190.9
	100	4	57614	60070	59079	881.7	57750	58342	58072	206.1	**57069**	57700	57396	198.8
	100	6	63177	64203	63659	407.4	62519	62954	62826	129.2	**61572**	62560	61874	305.4
	100	8	62226	64289	63197	705.5	62177	62858	62526	225.2	**61242**	62176	61653	316.9
	200	2	188236	189717	189088	540.7	187101	187840	187506	308.2	**184646**	186251	185399	645.7
	200	4	217089	226899	220829	3864.4	215050	217343	216133	865.8	**209997**	212572	211019	969.4
	200	6	223841	230953	227632	3013.5	221598	223228	222696	631.2	**219112**	220030	219396	367.8
	200	8	230311	232968	232118	1050.8	229676	231195	230561	573.1	**224404**	227783	226295	1385.4
	500	2	1163389	1172867	1167283	3898	1163284	1168799	1166500	2182.3	**1145288**	1150021	1147218	1866.8
	500	4	1254344	1266240	1261453	4444.3	1261893	1268112	1264735	2560.5	**1234250**	1243153	1238456	3627.6
	500	6	1307473	1314048	1311558	2625.7	1314401	1330141	1321302	6018.8	**1287421**	1294578	1290429	2731.4
	500	8	1313925	1325283	1320767	5135.4	1320614	1333508	1325290	5275	**1285338**	1297008	1289428	4784.4

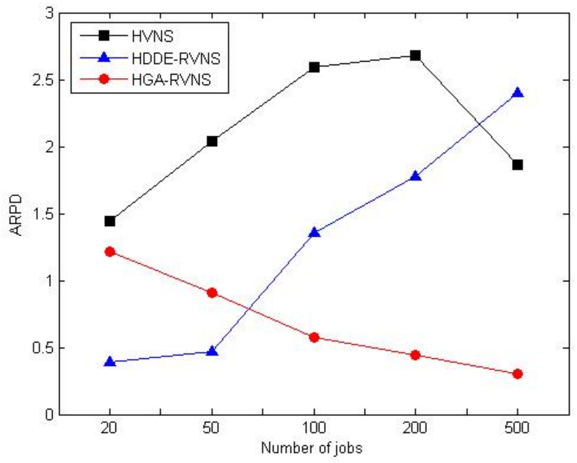

Fig. 20. Overall ARPD across *n*

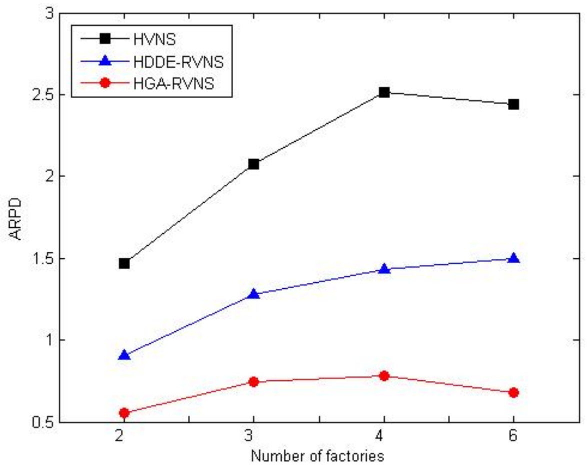

Fig. 21. Overall ARPD across *f*

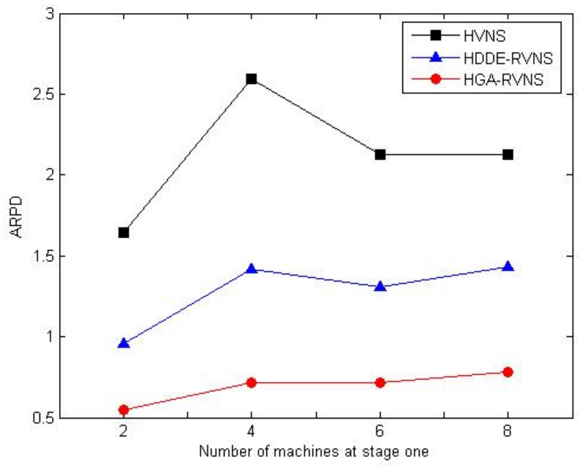

Fig. 22. Overall ARPD across *m*

Fig. 20 shows the performances of three proposed algorithms with respect to the number of jobs for ARPD. It can be seen from Fig. 20 that in average, in case of n = 20 and n = 50, HDDE-RVNS performs much better than HGA-RVNS and HVNS. In case of n = 100, 200 and n = 500, HGA-RVNS performs much better than HDDE-RVNS and HVNS.

Fig. 21 illustrates the performances of the three algorithms with respect to the number of factories. It is indicated that the performance of HGA-RVNS is not much effected by f, whereas the performance of HDDE-RVNS becomes slightly worse as f become larger. Both HDDE-RVNS and HGA-RVNS obtain much better overall ARPD across f than HVNS.

Fig. 22 shows the performances of the three proposed algorithms with respect to the numbers of machines at stage the first stage. It can be seen from the figure that the performances of the HGA-RVNS and the HDDE-RVNS are not much affected by m.

To verify the efficient of the local search step, the HGA and the HDDE without RVNS-based local search (HGA-NOV and HDDE-NOV) are also coded in Matlab7.0 on the same computer. The MinTCT of the five algorithms including HGA-NOV, HDDE-NOV, HGA-RVNS and HDDE-RVNS and HVNS with f = 4 are summarized in Table 6. The results with other f values were similar and not reported in the paper due to space limitation. It can be seen that both the proposed algorithms with local search perform much better than those without local search. Thus, it confirms the validity of the local search step in HDDE-RVNS and HGA-RVNS. It is clear that the proposed HGA-RVNS and HDDE-RVNS obtain a well balance between local search and global search. Also, we can see from the table that the proposed HVNS algorithm also outperforms the HGA-NOV and the HDDE-NOV.

Table 6
The best TCT obtained by the five algorithms when f = 4 (HGA-NOV, HGA-RVNS, HDDE-NOV, HGA-RVNS, and HVNS).

f	n	m	HDDE-NOV	HDDE-RVNS	HGA-NOV	HGA-RVNS	HVNS
			MIN	MIN	MIN	MIN	MIN
4	20	2	5229	**5150**	5254	5163	5163
	20	4	4518	**4433**	4478	4435	4454
	20	6	5629	**5497**	5607	5556	5566
	20	8	5384	**5303**	5451	5342	5348
	50	2	19814	**19299**	19922	19343	19494
	50	4	23866	**23041**	23819	23130	23428
	50	6	25399	**24424**	25047	24510	24882
	50	8	25888	**25104**	26780	25182	25436
	100	2	72552	70981	71978	**70347**	71419
	100	4	82608	80599	82399	**79933**	80851
	100	6	89683	87029	92727	**86697**	87419
	100	8	89938	86771	91494	**85387**	86891
	200	2	281250	270181	277174	**268367**	272141
	200	4	313311	307855	315274	**301485**	311066
	200	6	324445	319680	337881	**313469**	321322
	200	8	336496	329314	349973	**322592**	330639
	500	2	1737595	1708713	1759527	**1688839**	1705613
	500	4	1869055	1856815	1918338	**1811078**	1844266
	500	6	1960217	1920915	2028308	**1881212**	1917737
	500	8	1964768	1936557	2025591	**1885962**	1912951

5. Conclusions

In this paper, we addressed a novel distributed scheduling problem (DTSAFSP) with the *TCT* criterion where setup times are considered. Owning to the NP-hardness of the problem, A SPT-based algorithm and three hybrid meta-heuristics (HVNS, HDDE-RVNS, and HGA-RVNS) have been proposed.

At first, a comparison study among HVNS, HDDE-RVNS and HGA-RVNS has been conducted in order to evaluate their performances. Computational results showed that for small size instances (n = 20 and n = 50), HDDE-RVNS performs much better than HGA-RVNS and HVNS, whereas for large size instances (n = 100, n = 200 and n = 500), GA-RVNS performs much better than HVNS and HDDE-RVNS. It also showed that, the performances of HDDE-RVNS and HGA-RVNS are not much affected by the number of machines at the first stage and factories. Secondly, the effectiveness of the local search steps in HGA-RVNS and HDDE-RVNS has been examined. Computational results indicated that both the proposed algorithms achieve much better performance than HGA and HDDE without local search.

For future study, it will be interesting to develop more efficient meta-heuristics for the DTSAFSP-TCT. Also, consideration of due date related criteria will be useful in the DTSAFSP. Worthwhile extensions of our proposed algorithms for other novel DSPs are expected.

Acknowledgements

This work is supported in part by the National Natural Science Foundation of China under Grants 50975224, China Postdoctoral Science Foundation Funded Project (2012M521783), and Open Research Foundation from State Key Laboratory for Manufacturing Systems Engineering (Xi’an Jiaotong University)(SKLMS2012008).

References

[1] Lee CY, Cheng TCE, Lin BMT. Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. Management Science 1993; 39 (5): 616-25.

[2] Potts CN, Sevastjanov SV, Strusevich VA, Wassenhove LNV, Zwaneveld CM. The two stage assembly scheduling problem: complexity and approximation. Operations Research 1995; 43 (2): 346-55.

[3] Allahverdi A, Al-Anzi FS. A pso and a tabu search heuristics for assembly scheduling problem of the two stage distributed database application. Computers and Operations Research 2006; 33 (4): 1056-80.

[4] Koulamas C, Kyparisis GJ. The three-stage assembly flowshop scheduling problem. Computers and Operations Research 2001; 28(7): 689-704.

[5] Al-Anzi FS, Allahverdi A. A self-adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times. European Journal of Operational Research 2007; 182 (1): 80-94.

[6] Allahverdi A, Al-Anzi, FS. Evolutionary heuristics and an algorithm for the two- stage assembly scheduling problem to minimize makespan with setup times. International Journal of Production Research 2006; 44 (22): 4713-35.

[7] Mozdgir A, Fatemi Ghomi SMT, Jolai F, Navaei J. Two-stage assembly flow-shop scheduling problem with non-identical assembly machines considering setup times. International Journal of Production Research 2013; 51 (12): 3625-42.

[8] Tozkapan A, Kirca O, Chung CS. A branch and bound algorithm to minimize the total weighted flowtime for the two-stage assembly scheduling problem. Computers and Operations Research 2003; 30 (2): 309-20.

[9] Naderi B, Ruiz R. The distributed permutation flowshop scheduling problem. Computer and

Operations Research 2010; 37 (4): 754-768.

[10] Gao J, Chen R. A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. International Journal of Computational Intelligence Systems 2011; 4 (4): 497-508.

[11] Gao J, Chen R. and Deng W. An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. International Journal of Production Research 2013; 51 (3): 641-51.

[12] Wang SY, Wang L, Liu M, Xu Y. An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. International Journal of Production Economics 2013; 145 (1): 387-96.

[13] Jia HZ, Nee AYC, Fuh YH, Zhang YF. A modified genetic algorithm for distributed scheduling problems. Journal of Intelligent Manufacturing 2003; 14 (3-4): 351-62.

[14] Chan FTS, Chung, SH, Chan PLY. An adaptive genetic algorithm with dominated genes for distributed scheduling problems. Expert Systems with application 2005; 29 (2): 364-71.

[15] Chan FTS, Chung, SH, Chan PLY, Finke G, Tiwari MK. Solving distributed FMS scheduling problems subject to maintenance: Genetic algorithms approach. Robotics and Computer-Integrated Manufacturing 2006; 22 (5-6): 493-504.

[16] Giovanni LD, Pezzella F. An improved genetic algorithm for the distributed and flexible job-shop scheduling problem. European Journal of Operational Research 2010; 200 (2): 395-408.

[17] Allahverdi A, Al-Anzi, FS. The two-stage assembly scheduling problem to minimize total completion time with setup times. Computers and Operations Research 2009; 36(10): 2740-7.

[18] Sung CS, Kim HA. A two-stage multiple-machine assembly scheduling problem for minimizing sum of completion times. International Journal of Production Economics 2008; 113(2):1038-48.

[19] Allahverdi A. Two-machine flowshop scheduling problem to minimize total completion time with bounded setup and processing times. International Journal of Production Economics 2006; 103(1): 386-400.

[20] Allahverdi A., Gupta JND, Aldowaisan T. A review of scheduling research involving setup considerations, OMEGA The International Journal of Management Sciences, 1999; 27(2): 219-39

[21] Al-Anzi FS, Allahverdi A. A hybrid tabu search heuristic for the two-stage assembly scheduling problem. International Journal of Operations Research 2006; 3 (2): 109-19.

[22] Holland JH. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, Michigen; re-issured by MIT Press (1992) (1975).

[23] Eglese RW. Simulated annealing: A tool for operational research. European Journal of Operational Research 1990; 46 (3): 271-81.

[24] Storn R, Price K. Differential evolution - a simple and efficient heuristic for global optimization over continuous space. Journal of Global Optimization 1997; 11 (4): 341-59.

[25] Mladenovic N, Hansen P. Variable neighborhood search. Computer and Operations Research 1997; 24 (11): 1097-100.

[26] Glover F. Future paths for integer programming and links to artificial intelligence. Computers and Operations Research 1986; 13(5): 533-49.

[27] Dorigo M, Gambardella LM. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1997; 1(1):

53-66.

[28] Pinedo M. Scheduling: theory, algorithms and systems. Englewood cliffs, NJ: Prentice-Hall 2002.

[29] Hansen P, Mladenovic N. Variable neighborhood search: Principles and applications. European Journal of Operational Research 2001; 130 (3): 449-67.

[30] Lejeune MA. A variable neighborhood decomposition search method for supply chain management planning problems. European Journal of Operational Research 2006; 175 (2): 959-76.

[31] Liao CJ, Cheng CC. A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date. Computers and Industrial Engineering 2007; 52 (4): 404-13.

[32] Roshanaei V, Naderi B, Jolai F, Khalili M. A variable neighborhood search for job shop scheduling with set-up times to minimize makespan. Future Generation Computer Systems 2009, 25 (6): 654-61.

[33] Yazdani M, Amiri M, Zandieh M. Flexible job-shop scheduling with parallel variable neighborhood search algorithm. Expert Systems with Applications 2010; 37 (1): 678-87.

[34] Xing KY, Han LB, Zhou MC, Wang F. Deadlock-free genetic scheduling algorithm for automotated manufacturing systems based on deadlock control policy. IEEE Transactions on System, Man and Cybernetics, Part B: Cybernetics 2012; 42 (3): 603-15.

[35] Murata T, Ishibuchi H, Tanaka H. Genetic algorithms for flows shop scheduling problems. Computers & Industrial Engineering 1996; 30 (4): 1061-71.

[36] Storn R, Price K. Differential evolution - a simple and efficient heuristic for global optimization over continuous space. Technical Report TR-95-012, ICSI 1995.

[37] Montgomery DC. Design and analysis of experiments. Arizona: John Wiley and Sons 2005.

[38] Naderi B, Fatemi Ghomi SMT, Amninayeri M. A high performing metaheuristic for job shop scheduling with sequence-dependent setup times. Applied soft computing 2010; 10 (3): 703-10.

[39] Shokrollahpour E, Zandieh M, Dorri B. Group scheduling in flexible flow shops: a hybridised approach of imperialist competitive algorithm and electromagnetic-like mechanism. International Journal of Production Research 2011; 49 (16): 4965-77.

[40] Soltani R, Sadjadi SJ. Scheduling trucks in cross-docking systems: A robust meta-heuristics appoach. Transportation Research Part E 2010; 46 (5): 650-66.

[41] Wang L, Wang SY, Liu M. A Pareto-based estimation of distribution algorithm for the multi-objective flexible job-shop scheduling problem. International Journal of Production Research 2013; 51(12): 3574-92.