Expert Systems with Applications 41 (2014) 7754–7763

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Modeling and heuristics for scheduling of distributed job shops

B. Naderi[a,b], A. Azab[a,*]

[a] Product Lifecycle Management Research Lab, Department of Industrial and Manufacturing Systems Engineering, Faculty of Engineering, University of Windsor, Windsor, Canada
[b] Department of Industrial Engineering, Faculty of Engineering, University of Kharazmi, Karaj, Iran

ARTICLE INFO

Article history:
Available online 20 June 2014

Keywords:
Distributed job shop scheduling
Mathematical modeling
Greedy heuristics

ABSTRACT

This paper deals with the problem of distributed job shop scheduling in which the classical single-facility job shop is extended to the multi-facility one. The mathematical formulation of the problem is comprehensively discussed. Two different mixed integer linear programming models in form of sequence and position based variables are proposed. Using commercial software of CPLEX, the small sized problems are optimally solved. To solve large sized problems, besides adapting three well-known heuristics, three greedy heuristics are developed. The basic idea behind the developed heuristics is to iteratively insert operations (one at each iteration) into a sequence to build up a complete permutation of operations. The permutation scheme, although having several advantages, suffers from redundancy which is having many different permutations representing the same schedule. The issue is analyzed to recognize the redundant permutation. That improves efficiency of heuristics. Comprehensive experiments are conducted to evaluate the performance of the two models and the six heuristics. The results show sequence based model and greedy heuristics equipped with redundancy exclusion are effective for the problem.

1. Introduction

In the classical job shop, it is assumed that there is a single-facility with a set of m machines. There is also a set of n job and each job has its own process route among the machines. The problem is to schedule jobs on machines so as to minimize an objective which is always time-related. The most frequently used objective is makespan, i.e., the maximum completion time of jobs. Commonly, it is assumed that jobs are independent. Machines and jobs are all continuously available. The setup times can be ignored or included into processing times. A job can be processed by at most one machine at a time and a machine can process at most one job at a time.

Currently, we are facing quite some change with the structure and topology of small and midsize enterprises especially when it comes to the manufacturing world and metal cutting. Job shops are constantly facing more challenges and an increased need to reduce on both their costs and time-to-market. This has enforced a new decentralized scheme for the world of job shops, where shops in the high-wage developed countries started to establish branches for themselves in the low-wage developing world. This new paradigm of distributed manufacturing is increasingly replacing the traditional centralized single-facility one. It brings in advantages such as low production costs, production flexibility and enhanced overall manufacturing capabilities. Manufacturers become closer to both suppliers and customers, and thus are more able to adhere to local regulations and be more responsive to market changes. In DJS, we have a set of identical f facilities each of which consists of m machines. The problem of DJS is more complicated since two decisions have to be taken; first, the allocation of jobs to facilities and then production scheduling of jobs. It is additionally assumed that the job crossing is not allowed since it is very likely uneconomical or technologically difficult and impractical to transport a work-in-process job from one facility to anther for its remaining operations. In distributed scheduling, makespan minimization becomes the minimization of maximum makespan among facilities.

It is key when studying an optimization problem to develop its mathematical formulation. Besides being able to solve relatively small-to-midsize instances of the problem exact for optimality, however, we get also to rigorously and accurately define the problem. The problem is comprehensively formulates using two mixed integer linear programming (MILP) models. The first one treats the problem as a sequencing decision while the second oy67ne treats it as a positioning one. Computational complexities for both models have been presented and compared.

Due to inherent NP hardness of DSJ, the mathematical models are only capable of solving small to midrange problems for optimality. Hence, three well established heuristics are being adapted to the problem at hand; these are shortest processing time first (SPT), longest processing time first (LPT) and longest remaining

* Corresponding author. Tel.: +1 5192533000x4958; fax: +1 5199737062.
E-mail addresses: azab@uwindsor.ca, azab.ahmed@gmail.com (A. Azab).

http://dx.doi.org/10.1016/j.eswa.2014.06.023
0957-4174/© 2014 Elsevier Ltd. All rights reserved.

processing time (LRPT). Also, three greedy heuristics have been deployed. The algorithms are greedy since at each step, several alternatives are generated and the best one is being selected. The permutation encoding scheme is used to represent solutions. Although this encoding scheme has several advantages such as simplicity of encoding, less exhaustive computations, ease of adjustment to operators, and so forth, it suffers from a serious shortcoming, which is redundancy. That is, many different encoded solutions represent the same schedule. However, the algorithms have been treated to recognize and discard redundant permutations. Several numerical experiments are conducted to evaluate the performance of the two models and the six developed algorithms/heuristics as well as the efficiency of the developed redundancy mitigation mechanism. The results show sequence based model and greedy heuristics are effective for the problem.

The rest of the paper is organized as follows. Section 2 reviews the related literature. Section 3 proposes the mathematical models. Section 4 presents the adapted heuristics and proposed greedy algorithms. Section 5 conducts the numerical experiments. Finally, Section 6 concludes the paper and suggests few future research directions.

2. Literature review

In spite of the growing role of distributed and globalized manufacturing, the main focus in the literature on production scheduling is still so far on single-facility manufacturing. As of that dearth of the literature, Jia, Fuh, Nee, and Zhang (2002), Jia, Nee, Fuh, and Zhang (2003) study the distributed production scheduling problem and propose a genetic algorithm. Later on, in another paper, Jia, Fuh, Nee, and Zhang (2007) renew the previous genetic algorithm for the problem. This algorithm is a rather standard genetic algorithm that suffers from many serious shortcomings. For example, all individuals of a generation are of the same job-facility assignment. From a generation to another, all the job-facility assignment is changed together. Chan, Chung, and Chan (2005) propose a genetic algorithm for distributed flexible manufacturing systems. Chan, Chung, Chan, Finke, and Tiwari (2006) adapt this genetic algorithm for the same problem, however, considering this time with maintenance. Moreover, a memetic algorithm is applied to the same problem by Yadollahi and Rahmani (2009). Wang and Shen (2007) write a book about distributed manufacturing with a focus on planning and manufacturing problems rather than production scheduling. Behnamian and Fatemi Ghom (2013) consider distributed parallel machine problems and propose a genetic algorithm hybridized with local search.

Regarding distributed flow shop scheduling, more papers have been published. The first attempt to formulate this problem is done by Naderi and Ruiz (2010) where they develop six different mathematical models. They also propose different heuristics based on two job-facility assignment rules. After the pioneering paper of Naderi (2010), different metaheuristics have been applied to the problem. This is to include electromagnetism-like mechanism algorithm by Liu and Gao (2010), knowledge-based genetic algorithm by Gao, Chen, and Liu (2012), hybrid genetic algorithm by Gao and Chen (2011a), NEH based heuristic by Gao and Chen (2011b), variable neighborhood descent algorithm by Gao, Chen, Deng, and Liu (2012), tabu search by Gao, Chen, and Deng (2013), modified iterated greedy algorithm by Lin, Ying, and Huang (2013), estimation of distribution algorithm by Wang, Wang, Liu, and Xu (2013).

To the top of authors' knowledge, there is no paper directly studying the distributed job shop problem. Obviously, this paper represents the first attempt at mathematically modeling DJS allowing for a precise characterization of the problem. Apart from the mathematical modeling, some algorithms are proposed to effectively solve the problem.

3. Developed mathematical models

This section mathematically models the problem of scheduling distributed job shops by two different mixed integer linear programs. The application of integer programming models in solving scheduling problems starts with the early model of Wagner (1959). Yet, with earlier limitations on computing power and the lack of commercial software, the research progresses on this field were quite slow during the second half the past century. But, due to recent leaps in computing capacity and advent of specialized software, the MILP model and solution development is improving day after day. Even if this idea is accepted that mathematical models cannot be efficient solution algorithms, they are the first natural way to approach scheduling problems (Pan, 1997). It is the first natural step to describe a scheduling problem. Furthermore, mathematical models are along with in many solution methods such as branch and bound, dynamic programming and branch and price. The more efficient MILP models and its mechanism and conceptual workings, the better solution is obtained as results.

The following parameters and indices are used in both developed models:

n	the number of jobs $j, k = \{0, 1, 2, \ldots, n\}$, where job 0 is a dummy one to aid with defining and identifying the first job to be processed on a machine
m	the number of machine $i, l = \{1, 2, \ldots, m\}$
f	the number of facilities $r = \{1, 2, \ldots, f\}$
$p_{j,i}$	the processing time of job j on machine i
$a_{j,i,l}$	1 if machine i is used immediately after machine l in the processing route of job j and 0 otherwise
M	a large positive number

3.1. Operation-sequence based model

To formulate the problem, the first model views the problem as a sequencing decision. The decision variables are as such.

$X_{k,j,i,r}$	binary variable taking value 1 if job j is processed immediately after job k on machine i in facility r, and 0 otherwise (where $j \neq k$)
$S_{j,i}$	continuous variable for starting time of operation of job j on machine i

The MILP model is as follows.

$$\textit{Minimize } C_{max} \tag{1}$$

Subject to:

$$\sum_{k=0,k\neq j}^{n}\sum_{r=1}^{f} X_{k,j,1,r} = 1 \quad \forall_{j} \tag{2}$$

$$\sum_{k=0,k\neq j}^{n} X_{k,j,i,r} = \sum_{k=0,k\neq j}^{n} X_{k,j,1,r} \quad \forall_{j,i>1,r} \tag{3}$$

$$\sum_{j=1,j\neq k}^{n} X_{k,j,i,r} \leqslant \sum_{j=0,j\neq k}^{n} X_{j,k,1,r} \quad \forall_{k>0,i,r} \tag{4}$$

$$\sum_{j=1}^{n} X_{0,j,i,r} = 1 \quad \forall_{i,r} \tag{5}$$

$$\sum_{r=1}^{n} (X_{k,j,i,r} + X_{j,k,i,r}) \leqslant 1 \quad \forall_{j<n,k>j,i} \tag{6}$$

$$S_{j,i} \geqslant S_{j,l} + p_{j,l} \quad \forall_{j,i,l \neq i | a_{j,i,l}=1} \tag{7}$$

$$S_{j,i} \geqslant S_{k,i} + p_{k,i} - M(1 - X_{k,j,i,r}) \quad \forall_{k>0,j \neq k,i,r} \tag{8}$$

$$C_{max} \geqslant S_{j,i} + p_{j,i} \quad \forall_{j,i} \tag{9}$$

$$S_{j,i} \geqslant 0 \quad \forall_{k,i} \tag{10}$$

$$X_{k,j,i,r} \in \{0,1\} \quad \forall_{j,k \neq j,i,r} \tag{11}$$

Eq. (1) is objective function. Constraint set (2) determines the facility to which each job is assigned. Constraint sets (3) and (4) assure that all operations of a job are assigned to the same facility as successors and predecessors, respectively. Constraint set (5) determines the first job processed on each machine. Constraint set (6) is an optional set avoiding cross sequencing which means that a job cannot be successor and predecessor of another job. Constraint set (7) is to show that a job can be processed by at most one machine at a time while Constraint set (8) is to show a machine can process at most one job at a time. Constraint set (9) calculates the makespan. Constraint sets (9) and (10) define the decision variables.

3.2. Operation-position based model

The second model inspects the problem as positioning decisions. The decision variables are as such.

$X_{j,i,k}$	binary variable taking value 1 if job j occupies k-th position of machine i and 0 otherwise
$Y_{j,r}$	binary variable taking value 1 if job j is assigned to facility r, and 0 otherwise
$C_{j,i}$	continuous variable for completion time of operation of the job j on machine i

The second MILP model is as follows.

$$\text{Minimize } C_{max} \tag{12}$$

Subject to:

$$\sum_{k=1}^{n} X_{j,i,k} = 1 \quad \forall_{j,i} \tag{13}$$

$$\sum_{j=1}^{n} X_{j,i,k} = 1 \quad \forall_{i,k} \tag{14}$$

$$\sum_{j=1}^{f} Y_{j,r} = 1 \quad \forall_{j} \tag{15}$$

$$C_{j,i} \geqslant C_{h,i} + p_{j,i} - M\left(4 - X_{j,i,k} - \sum_{t=1}^{k-1} X_{h,i,t} - Y_{j,r} - Y_{h,r}\right) \quad \forall_{j,h \neq j,k>1,i,r} \tag{16}$$

$$C_{j,i} \geqslant p_{j,i} \quad \forall_{j,i} \tag{17}$$

$$C_{j,i} \geqslant C_{j,l} + p_{j,i} \quad \forall_{j,i,l | a_{j,i,l}=1} \tag{18}$$

$$C_{max} \geqslant C_{j,i} \quad \forall_{j,i} \tag{19}$$

$$C_{j,i} \geqslant 0 \quad \forall_{j,i} \tag{20}$$

$$Y_{j,r} \in \{0,1\} \quad \forall_{j,r} \tag{21}$$

$$X_{j,i,k} \in \{0,1\} \quad \forall_{j,i,k} \tag{22}$$

Eq. (12) is objective function. Constraint set (13) determines the position of each job on each machine. Constraint set (14) assures that each position on a machine is occupied by only one job. Constraint set (15) specifies the facility to which each job is assigned. Constraint set (16) is to show a machine can process at most one job at a time. Constraint set (17) is to show that the completion time of each operation is greater than its processing time. Constraint set (18) a job can be processed by at most one machine at a time. Constraint set (19) calculates the makespan. Constraint sets (20)–(22) define the decision variables.

4. The heuristics

In this section, three well-known heuristics are first adapted to DJS problem. To do this, a job-facility assignment rule is proposed. By this rule, jobs are assigned to facilities, and then sequenced by the heuristic. To more effectively solve the problem, three greedy heuristics are also proposed.

4.1. The three adapted rule-based heuristics (SPT, LPT and LRPT)

DJS includes two decisions: (1) job-facility assignment; (2) sequencing of jobs assigned to each facility. To apply well-known heuristics of job shops on DJS; therefore, a job-facility assignment rule is required. This decision is a tricky decision to make. The objective is to partition jobs into facilities so as to smooth the workload in different facilities.

One possibility is to consider the total processing time of each job j on different machines, i.e., $\sum_{i=1}^{m} p_{j,i}$ as the measure of workload. However, jobs might be assigned to facilities in such a way that they all have the same total processing times; yet, a good makespan might not be obtained. Matter of fact, jobs with large processing time on the very same machine when assigned to the same facility, makespan span increases quite significantly. Thus, this is not a proper strategy either. To better assign jobs to facilities, the workload on each machine is separately considered. That is, jobs are assigned to the facility where the maximum workload of machines is minimized. Only considering this criterion does not suffice still, since the processing route of jobs is not considered. Jobs with the same processing route might be assigned to the same facility, which increases makespan despite the fact workload is balanced on each machine on its own.

To take into consideration both factors (i.e., the processing times on each machine and processing routes), we use the notion workload, which is defined for each job j on each machine i as follows:

$$\text{Workload}\,(j,i) = \left(\sum_{k \in R_{j,i}} p_{j,k}\right) + p_{j,i} \quad \forall_{i,j}$$

where $R_{j,i}$ is the set of all machines preceding machine i in the processing of job j. In this case, if a machine is in last stages of the processing route has more adjusted workload comparing the achiness in early stages. Note that this new notion is used in the developed heuristics to ensure workload is balanced at large for each facility. To illustrate this concept, it is applied to a numerical example. Consider a DJS problem with $f = 2$, $n = 5$ and $m = 3$. The processing routes and times are shown in Table 1.

Table 1
The processing times and routes of the example.

Job	Machine			Processing route
	1	2	3	
1	5	12	3	{3, 2, 1}
2	8	3	7	{2, 1, 3}
3	2	7	3	{1, 3, 2}
4	1	5	9	{2, 1, 3}
5	8	2	14	{3, 1, 2}

Table 2
The workload of the example.

Job	Machine			Total	Rank
	1	2	3		
1	20	15	3	38	2
2	11	3	18	32	3
3	2	12	5	19	5
4	6	5	16	27	4
5	22	24	14	60	1

The Workload of each operation is shown in Table 2.

Regarding the total workloads, the jobs are ranked {5, 1, 2, 4, 3}. The two first jobs (jobs 5 and 1) are assigned to facilities 1 and 2, respectively. The workload of machines 1, 2, and 3 on the first facilities becomes 22, 24 and 14, respectively. The workload of machines 1, 2, and 3 on the second facility becomes 20, 15 and 3, respectively. Thus, the maximum workload in facilities 1 and 2 are 24 and 20, respectively. To assign the next job (job 2), the maximum workload is calculated if the job is assigned to a facility. Then, the job is assigned to the facility with minimum of the maximum workload. If job 2 is assigned to facility 1, the workloads become 33, 27 and 32 while if it is assigned to facility 2, we have workloads of 31, 18 and 21. The maximum workload of facilities 1 and 2 are 33 and 31, respectively. Therefore, job 2 is assigned to facility 2. To assign job 4, the workload in facility 1 becomes 28, 29 and 30 while in facility 2, the workload becomes 37, 23 and 37, respectively. Hence, the job is assigned to facility 1 with the maximum workload of 30. The procedure repeats for subsequent jobs.

Now, jobs assigned to each facility need to be sequenced. To do this, three well-known rules of SPT, LPT and LRPT are applied. They can be described as follows. In SPT, for each machine there is a list of jobs which are sorted in ascending order of processing times. Whenever a machine becomes available, the first job on its list is processed if the job is available and all of its preceding operations on rest of machines (according to job prescribed processing route) are already completed. If no such a job is available, the machine remains idle until such a job is available. The procedure of LPT is similar to SPT with a major difference. Unlike SPT, with the initial list corresponding to each machine, jobs are sorted in descending order of processing times.

It is necessary to indicate that schedules generated by heuristics are non-delay. A feasible schedule is called non-delay if no machine is kept idle while an operation is waiting for processing. In other words, requiring a schedule to be non-delay is equivalent to prohibiting unforced idleness in the schedules generated.

4.2. The proposed greedy heuristics

In this section three new high performing heuristics are proposed; they are construction heuristics that revolves around the insertion neighborhood concept that has been applied in a greedy fashion. This idea has shown high performance in many other scheduling problems (Dong, Huang, & Chen, 2008; Rad, Ruiz, & Boroojerdian, 2009). The proposed heuristics are implementations of insertion operators using a permutation representation.

A proper representation plays a key role in maintaining the search effectiveness. To encode DJS problem, multiple operation-based permutations, one for each facility is used. The permutation corresponding to a facility shows both jobs assigned to that facility and the relative order of the operations of those jobs on the machines. Since there are precedence constraints among the operations of each job, not all the permutations of the operations give feasible solutions. Therefore, the classical operation-based permutation is seen here as not effective and is not used.

In light of this fact, the following encoding scheme has been developed, with f operation permutations as explained before and a set of n_i operations corresponding to each job i. In this representation, each job index i occurs a number of times matching the number of operations it is composed of in the permutation of the facility to which the job is assigned. Hence, a job index will appear only in one of the f permutations. By scanning the permutation from left to right, the kth occurrence of a job index refers to the kth operation in the natural sequence of processing of operations of the job. Therefore, a permutation with repetition of job indices simply expresses the order in which the operations of the job are processed. As long as a job number is repeated as the number of its operations, the solution is always feasible.

To better illustrate the encoding scheme, it is applied to the previous example with 2 facilities, 5 jobs and 3 machines. Each job has 3 operations (Table 1). In this case, two permutations, one for each facility, are needed. One possible solution for this example is as such:

$$\begin{cases} 4,3,1,3,4,1,3,4,1 \\ 2,5,5,2,5,2 \end{cases}$$

In this encoded solution, job 1, 3 and 4 are assigned to facility 1, and jobs 2 and 5 to facility 2. The sequence of operations is determined by the permutations. In facility 1, the first job number is job 4, and its first operation is to visit machine 2 (as specified by its processing route). The earliest possible time to start this operation is time 0. The following appearance of job 4 in this permutation represents the second operation of job 4, and the next operation is to process job 3 on machine 1. Fig. 1 shows the resultant schedule of this encoded solution for facility 1. Makespan becomes 24. The same procedure is applied to decode the sequence of operations in facility 2. Finally, the general makespan is the maximum of these two makespans. Schedules generated using the employed coding scheme follow the semi-active class of schedules. A feasible schedule is called semi-active if no operation can be completed earlier without changing the order of processing on any one of the machines. In other words, the operations are processed as early as possible.

One shortcoming of this representation is redundancy. That is, different permutations might represent the same schedule. Consider the previous example where the encoded solution of operations in facility 1 is {4, 3, 1, 3, 4, 1, 3, 4, 1}. Another permutation {3, 4, 1, 3, 4, 1, 3, 4, 1}, where the positions of the two first job indices are swapped would also result in the same schedule as the former permutation (as shown in Fig. 1). This drawback wastes the computational time of algorithms by searching the redundant permutations (the same schedules). If the redundant permutations are known in advance, the algorithms can avoid searching them. In the developed heuristics, solutions generated are analyzed and redundant permutations are avoided. In this case, the search space is reduced and the efficiency of heuristics is increased.

Theorem 1. *Let θ' be a permutation produced from permutation θ by swapping two operations. Two permutations of θ' and θ are redundant if the two operations belong to the same job.*

Proof. Suppose θ' and θ represent the following permutations $\{\gamma_1, O_{j,l}, \gamma_2, O_{j,i}, \gamma_3\}$ and $\{\gamma_1, O_{j,i}, \gamma_2, O_{j,l}, \gamma_3\}$, respectively, where γ_1, γ_2 and γ_3 are the same operations for both θ and θ'. Only one of these two permutations is feasible. However, transforming the two permutations using the developed encoding scheme would result into the following: $\theta = \{\gamma_1, j, \gamma_2, j, \gamma_3\}$ and $\theta' = \{\gamma_1, j, \gamma_2, j, \gamma_3\}$, where both θ and θ' necessarily represent the same feasible schedule. Hence, it

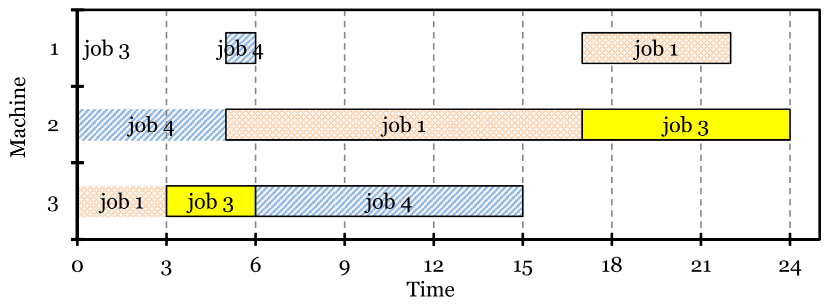

Fig. 1. The resultant schedule of the encoded solution of the example in facility 1.

becomes obvious how the usage of the developed coding scheme solves the feasibility problem. This completes the proof. □

Theorem 2. *Let θ' be a permutation produced from permutation θ by swapping two adjacent operations. Two permutations of θ' and θ are redundant if two different machines process these two operations.*

Proof. Suppose θ' and θ are $\{\gamma_1, O_{j,l}, O_{k,i}, \gamma_2\}$ and $\{\gamma_1, O_{k,i}, O_{j,l}, \gamma_2\}$, respectively. γ_1 and γ_2 are two sets of operations representing the rest of operations than $O_{j,l}$ and $O_{k,i}$; they are the same for both θ and θ'. Let us suppose the jobs of two operations $O_{j,l}$ and $O_{k,i}$ are different (i.e., $j \neq k$); otherwise, the two permutations are redundant (according to theorem 1). After scheduling the operations γ_1, suppose the completion time of machines and jobs are $(cm_1, \ldots, cm_l, \ldots, cm_i, \ldots, cm_m)$ and $(ct_1, \ldots, ct_j, \ldots, ct_k, \ldots, ct_n)$, respectively. Since operations γ_1 is the same for both θ' and θ, they have the same completion of jobs and machines. There are two cases.

Case 1. $l = i$ (i.e., $O_{j,l}$ and $O_{k,i}$ are processed on same machine)

After scheduling $O_{j,l}$ and $O_{k,i}$ according to θ, we have

$$cm : (cm_1, \ldots, cm_l + p_{j,l} + p_{k,l}, \ldots, cm_m)$$
$$ct : (ct_1, \ldots, ct_j + p_{j,l}, \ldots, ct_k + p_{j,l} + p_{k,l}, \ldots, ct_n)$$

and according to θ', we have

$$cm : (cm_1, \ldots, cm_l + p_{k,l} + p_{j,l}, \ldots, cm_m)$$
$$ct : (ct_1, \ldots, ct_j + p_{k,l} + p_{j,l}, \ldots, ct_k + p_{k,l}, \ldots, ct_n)$$

Therefore, they end up with different completion times. As a result, two permutations θ and θ' are two different schedules.

Case 2. $l \neq i$ (i.e., $O_{j,l}$ and $O_{k,i}$ are processed on different machines)

After scheduling $O_{j,l}$ and $O_{k,i}$ according to both θ and θ', the resulting completion time for the different machines (cm) and the different jobs (ct) are found to be the same as given below:

$$cm : (cm_1, \ldots, cm_l + p_{j,l}, \ldots, cm_i + p_{k,i}, \ldots, cm_m)$$
$$ct : (ct_1, \ldots, ct_j + p_{j,l}, \ldots, ct_k + p_{k,i}, \ldots, ct_n)$$

Since partial permutation γ_2 is the same for both θ and θ', the remaining operations yield the same schedule. Thus, two permutations θ and θ' are redundant. This completes the proof. □

4.2.1. Greedy heuristic $_1$ (GH$_1$)

In GH$_1$, two decisions of job-facility assignment and operation sequence are sequentially taken. Using the proposed job-facility assignment rule, the jobs are assigned to partition jobs into facilities. Then, operations of each facility are sequenced. To do that, GH$_1$ iteratively inserts operations, one at a time, into a sequence to build up a complete permutation of operations. The procedure can be described as follows: an initial random order of operations is built using its job indices where each job index appears as many as its operations (as discussed earlier). Then, job indices are one by one taken from the initial order and put into all the possible positions in sequence of scheduled job numbers. Finally, the best position is selected. The procedure repeats for the next job and so on and so forth.

Suppose a problem with $f = 2$, $n = 7$ and $m = 2$. Imagine jobs 2, 3 and 6 are assigned to facility 1. The initial random order of 6 operations is {2, 6, 3, 3, 6, 2}. At the first step, job 2 is taken. Since there is no scheduled operation, only one position exists. For the next job number, there are two positions. Therefore, we have two possible sequences:

$$\{2,6\} \quad \text{and} \quad \{6,2\}$$

The best sequence, among the possible ones, is selected. Assume that {2, 6} is the better one. To insert the next job index, there are three available positions and hence, three sequences.

$$\{3,2,6\}, \{2,3,6\} \quad \text{and} \quad \{2,6,3\}$$

Many of the resultant permutations are redundant, and the algorithm needs not check the schedules they represent. However, if the algorithm recognizes the redundancy in advance, it works much more efficiently. Using simple application of the two theorems above, the redundant permutations could be easily recognized and then discarded. Notice that each permutation is obtained by swapping two adjacent operations of the previous permutation. To check the redundancy, only these two operations need to be checked. If either of them belong to the same job or they are processed by two different machines, then the permutation is redundant and need not be checked. Fig. 2 shows the general outline of GH$_1$.

It is now evaluated how effective these theorems are. To do that, we count how many redundant permutations are avoided while implementing their straightforward application. The performance of GH$_1$ is calculated before and after avoiding redundant permutations later on in the experimental evaluation section.

4.2.2. Greedy heuristic$_2$ (GH$_2$)

This heuristic is also an insertion based heuristic. In GH$_1$, two decisions of job-facility assignment and sequencing are sequentially taken (i.e., no interaction between the two decisions). Unlike GH$_1$, job-facility assignment and sequencing are interactively determined in GH$_2$. The jobs are initially sorted. At each step, one job is taken from the initial order, allocated to a facility using rule-based assignment; only then sequencing of its operations starts.

To initially sort the jobs, they are arranged in descending order of their total processing times across all machines (i.e.,

```
Procedure: Greedy heuristic 1

Assign jobs to facilities (job-facility assignment)
For $k$=1 to $f$ do
    Determine random initial order of operations assigned to facility $k$
    For $i$=1 to $q$ do    %$q$ is the total number of operations assigned to facility $k$
        Take $i$th job number for the initial order
        For $h$=1 to $i$ do
            If the resultant sequence is not redundant do
                Test inserting the job number into $h$th position
            Endif
        Endfor
        Select the best
    Endfor
Endfor
```

Fig. 2. The general outline of GH_1.

$P_j = \sum_{i=1}^{m} p_{j,i}$). The first f jobs are taken and each is assigned to a facility. For each facility, a permutation is built by its corresponding job index. Then, next job in the initial order, is taken. To assign it to a facility, the facility with the lowest makespan is selected. To sequence its operations, the idea of iteratively inserting operations (one at each iteration) into a sequence to build up a complete permutation of operations has been utilized. The job index is added to the permutation of the selected facility for as many number as its operations. Each time, the job number is put into all the possible positions in sequence of scheduled job indices. Note that redundant sequences are discarded. Finally, the best position is selected. Fig. 3 shows the general outline of GH_2.

4.2.3. Greedy heuristic 3 (GH_3)

Like GH_2, this heuristic interactively takes two decisions of job-facility assignment and sequencing. The main difference is the job-facility selection rule. In GH_2, each job is assigned to the facility with the lowest makespan. While with GH_3, it performs in a greedier fashion and assign the job to the facility resulting in the lowest makespan after sequencing the operations of the job. That is, the assignments of jobs to all facilities are tested and the best one is selected. In this case, it is expected to end up with a better solution although more time-consuming. The other steps of GH_3 are similar to GH_2. The initial arrangement is in descending order of total processing times, etc.

5. Experimental evaluation

In this section, the performance of the two proposed mathematical models and six heuristics (SPT, LPT, TRPT, GH_1, GH_2 and GH_3) are evaluated and tested. To perform the evaluation, two experiments are conducted. In the first experiment, the models are compared in terms of both size and computational complexities. In the second experiment, the tested heuristics are compared with the optimal solution obtained by the model of the small instances. A set of larger instances is used to compare the algorithms for the performance, though.

These algorithms are implemented in Borland C++ and run on a PC with 3.4 GHz Intel® Core™ i7-3770 CPU and 8 GB of RAM memory. The performance measure used in this research is percentage deviation index (PDI). It can be calculated as follows.

$$\mathrm{PDI} = \frac{Alg - Min}{Max - Min} \times 100$$

where *Alg* is the makespan obtained by any of the algorithms. *Min* and *Max* are the lowest and largest makespans obtained for a given instance.

5.1. Data generation

To do the experiments mentioned above two sets of instances need to be generated. The first set is for the experiment with small

```
Procedure: Greedy heuristic 2

Arrange jobs in descending order of total processing times
For $j$=1 to $f$ do
    Take jth job in the initial order
    Assign the job to $j$th facility
Endfor
For $j$=f+1 to $n$ do
    Select the facility $k$ with the lowest makespan
    For $i$=1 to $n_j$ do    %$n_j$ is the number of operations of job $j$
        For $h$=1 to $q_k + i$ do %$q_k$ is the total number of operations assigned to facility $k$
            If the resultant sequence is not redundant do
                Test inserting the job number into $h$th position
            Endif
        Endfor
        Select the best
    Endfor
Endfor
```

Fig. 3. The general outline of GH_2.

Table 3
The considered levels for the data generation.

	The first set	The second set
f	{2, 3}	{2, 3, 4, 5}
n	{6, 8, 10, 12}	{15, 20, 30,50,100}
m	{3, 4}	{15, 20}

instances, the other one for the experiment with larger instances. The data for every one of the different instances consists of the number of facilities (f), the number of jobs (n), the number of machines (m) and the processing times ($p_{j,i}$). Table 3 shows the considered level for each set. For the first set, when $f = 2$, there are 8 combinations and when $f = 3$, only $n = 10$, 12 and $m = 3$, 4 are considered, summing up to 12 different combinations in total. For each combination, two instances are generated by random processing times taken from a uniform distribution between 1 and 99. For the second set, the instances of Taillard benchmark for job shops (Taillard, 1993) are being used. This benchmark includes 8 combinations for n and m, and 10 instances for each combination. It sums up to 80 instances. Each instance is solved by different levels of f; thus, there are 320 instances.

5.2. Models evaluation

In order to evaluate the developed MILP models, there are two frequently used performance measures: size and computational complexities (Pan, 1997; Stafford & Tseng, 2002). The size complexity is the numbers of binary variables (BV), the number of continuous variables (CV) and the number of constraints required by a model (Pan, 1997). The computational complexity is the time required by a model to solve an instance (Stafford & Tseng, 2002).

Table 4 shows the size complexity of the two models to formulate a problem with n jobs, m machines and f facilities. Regarding binary variables, size complexity of both models are $O(n^2)$. Thus, they both are quadratic in n. They are linear in both m and f, however. Although having the same complexity, the position-based model quantitatively needs fewer BVs. Both models require the same number of CVs. Regarding constraints, the sequence-based model is $O(n^2)$ (i.e., it is quadratic in n) while the position-based model is $O(n^3)$ (i.e., it is cubic in n). That is, its size complexity is greater than that of the sequence-based model. Considering m and f, the models are linear. The first conclusion is the sequence based model seems to be more effective since it has lower size complexity in the number of constraints. Moreover, it is clear from the analysis, the most influential parameter on the size complexity of the model is n. The other parameters m and f have only linear effect on the size complexity.

To have a numerical view, Table 5 shows the number of BVs, CVs and constraints for different example sizes. The position model requires less number of binary variables. For instance, when $n = 40$, $m = 10$ and $f = 5$, sequence-based model needs 80,000 BVs while the position-based model generates 16,200 BVs. Both models relatively have few CVs. Regarding the number of constraints, the sequence-based model needs far less constraints than the

Table 4
The size complexity of the two models.

Factor (No. of)	Model	
	Sequence-based	Position-based
Binary variables	n^2mf	$n^2m + nf$
Continuous variables	nm	nm
Constraints	$n^2m(f+\frac{1}{2})+n(\frac{3}{2}m+1)+mf(n+1)$	$4nm + n(f + 1) + nmf(n - 1)^2$

Table 5
Number of binary variables comparison of the models.

Factor (No. of)	Problem size			Models	
	n	m	f	Sequence	Position
Binary variables					
	10	5	2	1000	520
	20	5	3	6000	2060
	30	10	4	36,000	9120
	40	10	5	80,000	16,200
Continuous variables					
	10	5	2	50	50
	20	5	3	100	100
	30	10	4	300	300
	40	10	5	400	400
Constraints					
	10	5	2	1445	8330
	20	5	3	7485	108,780
	30	10	4	42,220	1,010,550
	40	10	5	90,690	30,438,840

position-based model does. The difference becomes more significant when the problem size grows up.

To analyze the computational complexity of the models, we use the very same set of data. The instances of this set are solved by the implementation of MILP models in CPLEX 12. The models are given a maximum time limit of 3000 seconds. The sequence-based MILP model optimally solves 17 instances out of 24 ones, while the position-based model solves only 7 instances. The sequence-based model solves all the instances up to 10 jobs. It also solves some of instances with 12 jobs. The position-based model optimally solves instances with 6 jobs. It rarely solves instances with 8 and 10 jobs. Considering the computational complexity, the sequence-based model outperforms the position-based model (see Table 6).

5.3. Heuristics evaluation

To evaluate the performance of the algorithms, we use both data sets described earlier. We first evaluate the general performance of the six tested algorithms vs. the optimal solution of the

Table 6
The results of the models.

n	m	f	Sequence		Position	
			C_{max}	Time (sec)/ optimality gap (%)	C_{max}	Time (sec)/ optimality gap (%)
6	3	2	**25**[a]	0.01	**25**[a]	0.13
6	3	2	**28**[a]	0.88	**28**[a]	14
6	4	2	**32**[a]	0.49	**32**[a]	14.16
6	4	2	**28**[a]	0.56	**28**[a]	4.8
8	3	2	**28**[a]	6.77	**28**[a]	260
8	3	2	**29**[a]	14.88	29	17%
8	4	2	**37**[a]	28.39	37	8%
8	4	2	**40**[a]	11.33	40	7%
10	3	2	**33**[a]	1132	33	30%
10	3	2	32	27%	33	30%
10	4	2	**28**[a]	276	28	25%
10	4	2	38	2%	42	30%
10	3	3	**26**[a]	6.64	**26**[a]	10.05
10	3	3	**30**[a]	88.08	30	20%
10	4	3	**35**[a]	23.59	35	2.80%
10	4	3	**33**[a]	11.86	**33**[a]	28
12	3	2	40	37%	40	37%
12	3	2	37	24%	42	33%
12	4	2	40	17%	46	39%
12	4	2	44	11%	50	38%
12	3	3	**30**[a]	2638	32	25%
12	3	3	**29**[a]	2366	30	6.70%
12	4	3	36	3%	39	17.90%
12	4	3	**33**[a]	603	34	5.90%

[a] Optimal solution.

Table 7
The results of the experiment with small instances.

n	m	f	Optimal $C_{\max}$	Algorithms					
				GH_1	GH_2	GH_3	LPT	SPT	LRPT
6	3	2	25	25	25	**25**	27	25	25
6	3	2	28	35	33	**31**	40	34	35
6	4	2	32	35	40	**32**	35	35	35
6	4	2	28	33	30	**29**	30	33	33
8	3	2	28	37	**28**	30	37	44	49
8	3	2	29	**30**	31	31	37	31	39
8	4	2	37	44	39	**39**	56	**39**	46
8	4	2	40	46	44	42	52	**41**	59
10	3	2	33	42	40	**35**	58	47	48
10	4	2	28	33	32	**30**	42	33	34
10	3	3	26	27	27	**26**	29	36	36
10	3	3	30	33	35	**31**	40	33	33
10	4	3	35	44	41	**38**	43	51	44
10	4	3	33	39	37	**34**	42	39	41
12	3	3	30	34	35	**33**	40	39	38
12	3	3	29	35	36	**32**	42	43	43
12	4	3	33	37	41	**34**	42	42	51
Average optimality gap				**15.9**	**13.1**	**5.3**	**31.4**	**23.4**	**31.1**

first data set obtained by the models. Table 7 shows the results of these 17 instances. Among the algorithms, GH_3 performs best by an average optimality gap of 5.3%. Moreover, it results in the lowest makespan in 14 instances out of 17 ones.

Now, we compare the tested algorithms on the second data set of the larger instances. All its 320 instances are solved by the algorithms. Table 8 shows the results averaged by the combinations of (n, m). The proposed GH_3 outperforms the other heuristics by far. It provides the average PDI of 1.21%. The second best heuristic is GH_1 with average PDI of 37.83%. GH_2 and SPT perform almost similarly. The worst performing heuristic is LRPT with average PDI of 87.93%. Note that the difference between GH_2 and GH_3 is in job-facility assignment and this shows how important the job-facility assignment rule is.

To further analyze the results, we use ANOVA (Analysis of Variance) and LSD (Least Significant Differences) tests. Fig. 4 shows the results of the experiments. As it could be seen GH_3 statistically outperforms the other algorithms. GH_1 is statistically the second best. GH_2 and SPT are statistically similar.

Fig. 5 shows the average PDI of the tested algorithms vs. the number of jobs. The proposed GH_3 provides the best performance in all the levels of the number of jobs. There is a clear trend that SPT performs better in larger sizes while LPT and LRPT work worse in larger sizes. Fig. 6 shows the average PDI of the tested algorithms vs. the number of facilities. As expected, the gap between GH_3 and the other algorithms increases when the number of facilities increases. GH_2 performs better than SPT in larger sizes. A clear

Table 8
The results of the algorithms on the large instances.

n	m	Algorithms					
		GH_1	GH_2	GH_3	LPT	SPT	LRPT
15	15	37.65	55.30	1.21	84.61	75.96	85.30
20	15	35.56	56.12	2.19	79.01	71.09	87.15
	20	38.54	60.24	1.13	84.29	82.22	87.28
30	15	48.65	62.63	0.16	81.17	61.46	89.36
	20	38.24	71.24	3.07	68.67	66.07	76.27
50	15	32.32	43.35	0.43	77.15	43.70	95.43
	20	40.13	70.27	2.84	66.65	44.68	90.95
100	20	31.54	54.95	2.12	82.99	32.75	91.67
Average		37.83	59.27	1.64	78.07	59.74	87.93

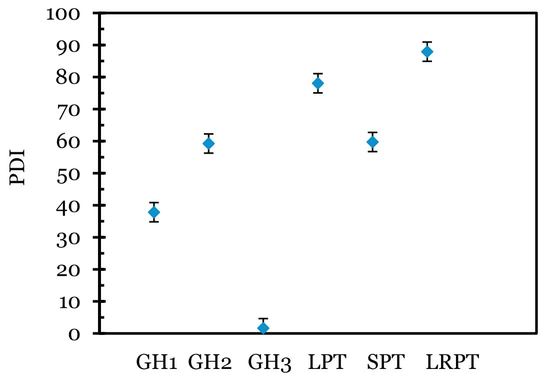

Fig. 4. The average PDI and LSD intervals of the tested algorithms.

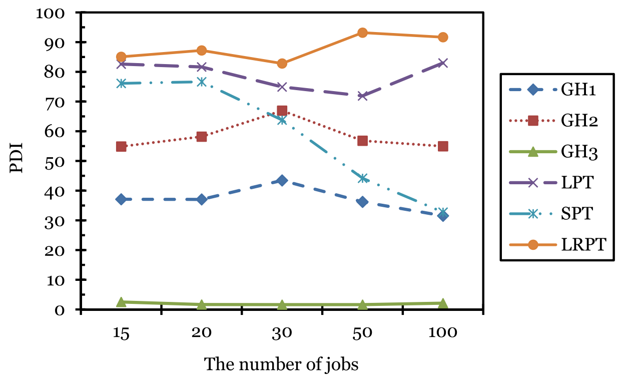

Fig. 5. The average RPD of the tested algorithms vs. the number of jobs.

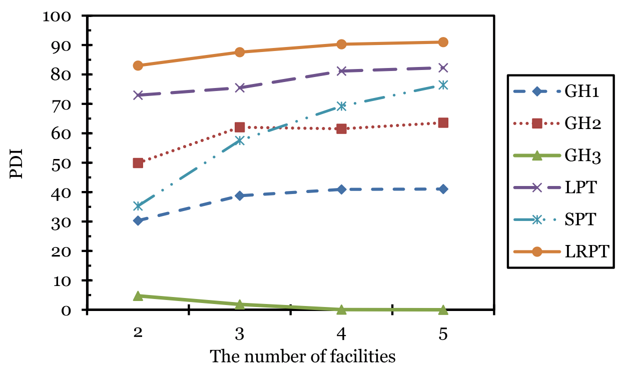

Fig. 6. The average RPD of the tested algorithms vs. the number of workstations.

trend shows: "the larger the number of facilities is, the worse the SPT performs".

5.4. Evaluation of computational time and redundancy exclusion

To this end, the computational time elapsed by the algorithms is reported. Table 9 shows the time averaged by the number of jobs. Generally, all the heuristics are very efficient since they solve instances with 100 jobs only in less than one second. To be more precise, LPT, SPT and LRPT are simple rules, and their computational time is less than 0.01 second. Comparing the greedy heuristics, GH_2 is faster than the two others.

Now, we analyze how efficient redundancy exclusion is. It is important to remind the reader that the algorithms discard the redundant permutations by applications of the proposed research

Table 9
The computational time of the tested algorithms (sec).

n	Algorithms			
	GH_1	GH_2	GH_3	LPT/SPT/LRPT
15	0.0008	0.0012	0.0035	0.0004
20	0.0043	0.0033	0.0086	0.0008
30	0.0127	0.0084	0.0252	0.0010
50	0.0609	0.0307	0.0982	0.0008
100	0.5953	0.2809	0.9984	0.0020

Table 10
The performance of GH_3 with redundancy exclusion and inclusion.

n	Computational time (sec)			No. of permutations checked		
	Exclusion	Inclusion	Δ	Exclusion	Inclusion	Δ
15	0.0035	0.0129	%72.8	8636	24,527	%64.8
20	0.0086	0.0414	%79.2	20,862	57,341	%63.6
30	0.0252	0.1297	%80.6	46,359	127,256	%63.6
50	0.0982	0.5879	%83.3	127,597	340,136	%62.5
100	0.9984	7.0359	%85.8	645,615	1714,351	%62.3
Average			%80.3			%63.4

finding. The data in Table 9 show the computational time of the proposed heuristics equipped with the application of redundancy exclusion properties. In Table 10, the performance of GH_3 in the two states is compared, one with exclusion of redundant permutations from the search and one with its inclusion. To this end, we show both computational time and the number of permutations checked in the two states. The percentage of reduction is also shown by column Δ.

As it can be seen, the computational time is interestingly reduced by more than 80%. For example, the average computational time to solve an instance with 100 jobs with exclusion of redundant permutations is less than 1 second while without it, it becomes more than 7 s. Regarding the number of permutations to be checked, it is cut down by more than 63%. For example, while solving an instance with 20 jobs, we need to check 57,341 different permutations. While among these permutations, only 20,862 permutations are not redundant and need to be checked. That is, 36,479 permutations are recognized as redundant permutations and need not be checked.

To have a more detailed picture, Figs. 7 and 8 show the percentage of computational time and permutations reductions vs. the number of jobs, machines and facilities, respectively. In Fig. 7, we can see that the computation time reduction is more significant in larger number of jobs and machines. While for larger number of facilities, it is decreased. This was expected, since the larger number of facilities is, the less average number of jobs assigned to each facility is. In Fig. 8, when the number of jobs and machines is increased, the percentage of reduction is reduced. The opposite trend is viewed when the number of facilities is increased.

6. Conclusion and future research directions

Industrial facilities are merging to a distributed facility to be closer to their customers, to comply with the local laws, to produce and market their products more effectively, and to be responsive to market changes more quickly. Since the globalization is vital to industries to survive, facilities require methodologies to effectively schedule their jobs in a distributed manner. Considering the complexity of scheduling in such a system, particular care must be taken to develop a specialized models and algorithms for the problem. The available models and algorithms on job shops fail to consider the distributed nature and aspect of the problem. Thus, the

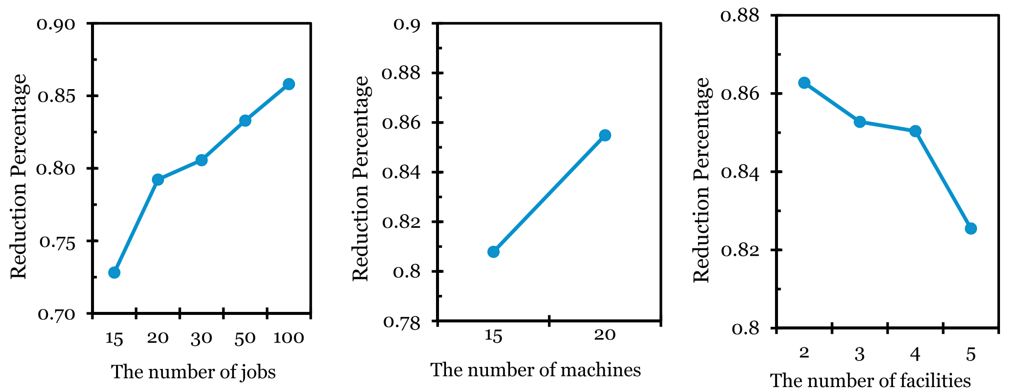

Fig. 7. The percentage of time reduction vs. problem sizes.

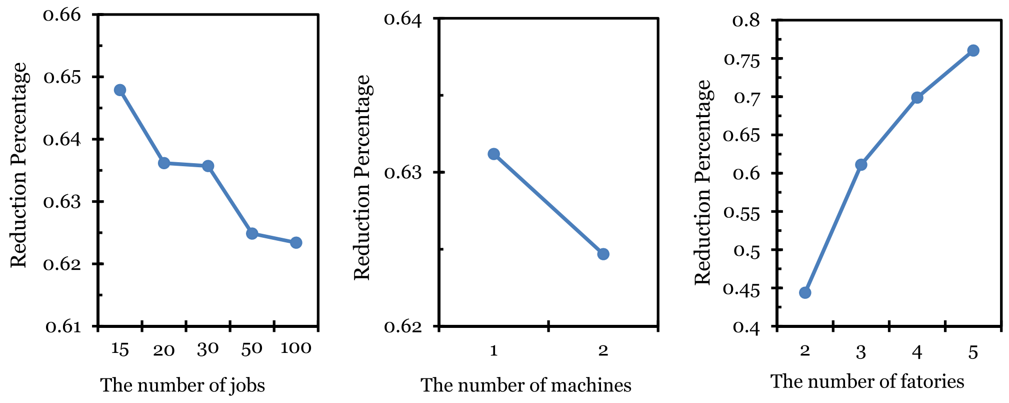

Fig. 8. The percentage of permutation reduction vs. problem sizes.

developed mathematical models and solution algorithm can be applied by industries to approach the modern configuration of distributed systems.

Considering the trend of globalization, there is no research papers that directly study the distributed job shop problem. In this paper, two mathematical models were first developed for distributed job shops since the problem has not been mathematically modeled. The two models in form of mixed integer linear programs with sequence and position based variables. Using the mathematical models, the problem was formulated and solved exact. The models optimally solve small sized problems using CPLEX. The models developed were evaluated with regards to their computational complexities. It turned out that the sequence-based model is more effective since it has less number of constraints. However, both models have same number of binary and continuous variables. The most influential parameter on the size complexity of the models is the number of jobs. The sequence and position-based models solve all the instances up to 12 jobs and 8 jobs, respectively. Therefore, it could be concluded that the sequence-based model outperforms the position-based model.

Since the problem is NP-hard, to solve the problem, utilization of heuristics was inevitable. Three well-known heuristics were first adapted for the problem. The heuristics were the shortest processing times first, longest processing time first and total remaining processing time. DJS includes two decisions of job-facility assignment and sequencing of jobs assigned to each facility. To apply well-known heuristics of job shops on DJS, it was important to develop a job-facility assignment rule. Furthermore, three greedy heuristics were developed. The basic idea is to iteratively insert operations (one at each iteration) into a sequence to build up a complete permutation of operations in a constructive manner. The permutation scheme, although having several advantages, suffers from redundancy; i.e., many different permutations represent the very same schedule. This issue was analyzed and redundant permutations were checked and eliminated. The developed heuristics were evaluated against the exact optimal solutions obtained using mathematical programming and the developed formulations. They were also compared against the set of instances taken from Taillard benchmark. With regards to the optimal solutions obtained, it is concluded that GH_3 performs best with an average optimality gap of 5.3%. It results in the lowest makespan in 14 instances out of 17 ones. The proposed GH_3 highly outperforms the other heuristics with an average PDI of 1.21%. The second best heuristic is GH_1 with an average PDI of 37.83%. Finally, the redundancy mitigation mechanisms were also scrutinized. The results showed that more than 80% reduction in computational time with simple application of redundancy exclusion theorems.

For future work, it will be interesting to look at distributed job shops where facilities are not identical since it is rarely the case when an enterprise has the exact multiple identical facility duplicated. More precisely, each facility has machines of different technologies. Also, it will be interesting to extend the problem and look at the transportation time from facility to customers. Each job is the order of a customer; therefore, it is imperative that its distance to facility impacts the decision of assignment of jobs to facilities. Since the greedy heuristics developed are constructive ones, it could also be added value if new iterative metaheuristics such as particle swarm optimizations and artificial immune algorithms could be further explored.

Acknowledgments

The authors like to thank the natural sciences and engineering research council of Canada (NSERC) for contributing to funding this research. An NSERC Discovery grant #811008.

References

Behnamian, J., & Fatemi Ghom, S. M. T. (2013). The heterogeneous multi-facility production network scheduling with adaptive communication policy and parallel machine. *Information Sciences, 219*, 181–196.

Chan, F. T. S., Chung, S. H., & Chan, P. L. Y. (2005). An adaptive genetic algorithm with dominated genes for distributed scheduling problems. *Expert Systems with Applications, 29*(2), 364–371.

Chan, F. T. S., Chung, S. H., Chan, L. Y., Finke, G., & Tiwari, M. K. (2006). Solving distributed FMS scheduling problems subject to maintenance: Genetic algorithms approach. *Robotics and Computer-Integrated Manufacturing, 22*(5–6), 493–504.

Dong, X., Huang, H., & Chen, P. (2008). An improved NEH-based heuristic for the permutation flowshop problem. *Computers & Operations Research, 35*, 3962–3968.

Gao, J., & Chen, R. (2011a). A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Computational Intelligence Systems, 4*(4), 497–508.

Gao, J., & Chen, R. (2011b). An NEH-based heuristic algorithm for distributed permutation flowshop scheduling problems. *Scientific Research and Essays, 6*(14), 3094–3100.

Gao, J., Chen, R., & Deng, W. (2013). An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research, 51*(3), 641–651.

Gao, J., Chen, R., Deng, W., & Liu, Y. (2012). Solving multi-facility flowshop problems with a novel variable neighbourhood descent algorithm. *Journal of Computational Information Systems, 8*(5), 2025–2032.

Gao, J., Chen, R., & Liu, Y. (2012). A knowledge-based genetic algorithm for permutation flowshop scheduling problems with multiple facilities. *International Journal of Advancements in Computing Technology, 4*(7), 121–129.

Jia, H. Z., Fuh, J. Y. H., Nee, A. Y. C., & Zhang, Y. F. (2002). Web-based multi-functional scheduling system for a distributed manufacturing environment. *Concurrent Engineering – Research and Applications, 10*(1), 27–39.

Jia, H. Z., Fuh, J. Y. H., Nee, A. Y. C., & Zhang, Y. F. (2007). Integration of genetic algorithm and Gantt chart for job shop scheduling in distributed manufacturing systems. *Computers and Industrial Engineering, 53*(2), 313–320.

Jia, H. Z., Nee, A. Y. C., Fuh, J. Y. H., & Zhang, Y. F. (2003). A modified genetic algorithm for distributed scheduling problems. *Journal of Intelligent Manufacturing, 14*(3–4), 351–362.

Lin, S. W., Ying, K. C., & Huang, C. Y. (2013). Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. *International Journal of Production Research, 51*(16), 5029–5038. http://dx.doi.org/10.1080/00207543.2013.790571.

Liu, H., & Gao, L. (2010). A discrete electromagnetism-like mechanism algorithm for solving distributed permutation flowshop scheduling problem. In *IEEE international conference on manufacturing automation*. http://dx.doi.org/10.1109/ICMA.2010.17.

Naderi, B., & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers and Operations Research, 37*, 754–768.

Pan, C. H. (1997). A study of integer programming formulations for scheduling problems. *International Journal of Systems Science, 28*, 33–41.

Rad, S. F., Ruiz, R., & Boroojerdian, N. (2009). New high performing heuristics for minimizing makespan inpermutation flowshops. *Omega, 37*, 331–345.

Stafford, E. F., & Tseng, F. T. (2002). Two models for a family of flowshop sequencing problems. *European Journal of Operational Research, 142*, 282–293.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research, 64*, 278–285.

Wagner, H. M. (1959). An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly, 6*(2), 131–140.

Wang, L., & Shen, W. (2007). *Process planning and scheduling for distributed manufacturing*. London: Springer.

Wang, S., Wang, L., Liu, M., & Xu, Y. (2013). An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. *International Journal of Production Economics, 145*(1), 387–396. http://dx.doi.org/10.1016/j.ijpe.2013.05.004i.

Yadollahi, M., & Rahmani, A. M. (2009). Solving distributed flexible manufacturing systems scheduling problems subject to maintenance: Memetic algorithms approach. In *IEEE ninth international conference on computer and information technology*. http://dx.doi.org/10.1109/CIT.2009.113.