Int J Adv Manuf Technol (2012) 62:1249–1260
DOI 10.1007/s00170-011-3859-4

ORIGINAL ARTICLE

A hybrid metaheuristic for concurrent layout and scheduling problem in a job shop environment

Mohammad Ranjbar · Mojtaba Najafian Razavi

Received: 27 August 2010 / Accepted: 12 December 2011 / Published online: 28 December 2011

Abstract In order to obtain efficiency and flexibility, assignment of machines' layout and determination of jobs' schedule on each machine are among the most important decisions. These decisions are interrelated and may impact each other but they are often treated separately or as a sequential decision in prior researches. In this paper, we propose a new approach to concurrently make the layout and scheduling decisions in a job shop environment. In other words, we consider an extension of the well-known job shop scheduling problem with transportation delay in which in addition to decisions made in the classic problem, the locations of machines have to be selected among possible sites. The only goal of the problem is the minimization of the makespan. A hybrid metaheuristic approach based on the scatter search algorithm is developed to tackle this problem. Using 43 randomly generated benchmark instances, the performance of the scatter search and its components are evaluated. We also applied our procedure to the classic job shop scheduling problems. Computational results show that our procedure is efficient.

Keywords Job shop scheduling problem · Layout · Metaheuristic · Scatter search

M. Ranjbar (✉)
Department of Industrial Engineering, Faculty of Engineering,
Ferdowsi University of Mashhad,
Mashhad, P.O. Box: 91775-1111, Iran
e-mail: m_ranjbar@um.ac.ir

M. N. Razavi
Imam Reza University,
Mashhad, Iran

1 Introduction

The design of an effective and flexible job shop production system in some companies involves two major decisions: determination of layout and scheduling. Layout involves the process of assignment of machines to possible locations and scheduling deals with the problem of sequencing different jobs on each machine. These decisions are interrelated and may impact each other but they are often treated separately or as a sequential decision in prior researches.

It is often the case that growing companies need to purchase some new machines, produce new products or increase production quantity. In these cases, optimization of layout and scheduling is proposed. In practice, usually these two problems are tackled separately, while sometimes their impacts on each other are not negligible. A similar situation happens in companies having seasonal production. As a practical example, consider a factory that produces some types of air cooler in spring and summer and some types of gas stoves in fall and winter. Although there are several common machines and similar operations in production of these two categories of products, some machines and operations are different. Basically, scheduling of operations for production of air cooler based on the optimal layout designed for production of gas stove or vice versa is not an optimal decision. In addition, assignment of machines to possible locations regardless to operations and their routes increases transportation costs.

The problem proposed in this paper is the concurrent layout and scheduling problem in a job shop environment (CLSPJE). In the CLSPJE, it is assumed that there are a set of jobs where each one includes a set of operations. Also, there are a set of machines and a set of possible locations where the number of machines and locations are identical. For simplicity, the assumption of the quadratic assignment problem (QAP) is considered where each machine can be assigned to every possible site. It is presumed that at the beginning, all jobs are in a fixed place like a warehouse. Moreover, all of the classic job shop scheduling problem (JSP) assumptions are considered that involves the non-preemptive scheduling of each job's operations on machines in order to minimize the makespan. It is assumed that the sequence of operations for each job is predetermined, neither release times nor due dates are specified, each machine can process only one job at a time and each job can be processed on only one machine at a time. In addition to these assumptions, it is supposed that transportation times between each pair of machines as well as between the warehouse and each machine is symmetric. The transportation times are job- and machine–independent, and they depend only on distances. Moreover, different jobs can be transported simultaneously and speed of transportation is identical for all jobs. The only goal of the problem is minimization of the makespan. Since the JSP is NP-complete [1] and this problem is harder than the JSP, it is NP-complete as well.

The remainder of this paper is organized as follows. A review of the literature is given in Section "2". Section "3" describes the problem description and formulation. The schedule representation and generation scheme are discussed in Section "4". Also, the scatter search algorithm is described with detail in Section "5". Computational results are presented in Section "6", while Section "7" is reserved for overall conclusions and suggestions for future research.

2 Literature review

The literature on the concurrent scheduling and layout problem is almost void but there are some related works on the QAP and the JSP. Several exact and heuristic procedures are presented in the literature for the QAP. Branch-and-bound [2] and branch-and-cut [3] are the exact algorithms that have been applied for the QAP. Also, a cooperative parallel tabu search [4], a robust tabu search [5], a greedy genetic algorithm [6] and a hybrid genetic algorithm [7] are some of the recent metaheuristics designed for this problem. In addition, an iterated local search [8] and a novel chaotic search [9], belonging to the heuristic approaches, have been developed for the QAP.

On the other hand, many researchers have mainly focused on solving the JSP. The JSP is widely acknowledged as one of the most difficult NP-complete problems [1]. Thus, it is unrealistic to solve even a medium-sized problem using time-consuming exact algorithms such as integer programming [10] or branch-and-bound [11]. Therefore, metaheuristic approaches such as tabu search [12–14], hybrid tabu search [15], simulated annealing [16], scatter search [17], filter-and-fan [18], ant colony [19] and particle swarm optimization [20] have been developed for the JSP and achieved very good results. Moreover, heuristic approaches like those in Ref. [21], developed based on global equilibrium search techniques, and Ref. [22], developed based upon utilizing the properties of backbone and big valley, have found very good results for the hard instances of the JSP.

The JSP with transportation delay has been considered by Hurink and Knust [23], but in which the location of each machine is fixed. Also, a dynamic scheduling approach for the spatial layout planning for block assembly in the shipbuilding industry was proposed by Li et al. [24]. Spatial scheduling problem, in general, consists of the spatial layout planning and the spatial sequencing of objects to be scheduled. Moreover, in the field of cellular manufacturing, a genetic algorithm has been developed by Wu et al. [25] for integrating cell formation with machine layout and scheduling.

3 Problem description and formulation

The CLSPJE can be described based on the following list of assumptions:

a. There are n jobs, m machines, m possible sites and a warehouse.
b. Each job i where i=1,…, n constituted of a set of operations (k, i) where k indicates index of machines. We do not consider recirculation, i.e., each job visits each machine at most one time. This assumption is considered for simplicity of notations and is based on the formulation of Pinedo [26].
c. Each job i has a specified route on the machines. This is denoted by set A and includes the precedence constraints among operations of each job.
d. The processing of job i lasts for p_{ki} time units on machine k where k=0 indicates the warehouse and p_{oi}=0 for i=1,…, n.
e. Transportation time between locations u and v is denoted by t_{uv}.

Based upon the assumptions of the CLSPJE mentioned in Section "1", it can be formulated as an integer programming model by introducing the following variables:

s_{ki} = the start time of job i on machine k

$$z_{ku} = \begin{cases} 1;\text{if machine } k \text{ is placed on location } u \\ 0;\text{otherwise} \end{cases}$$

$$x_{kij} = \begin{cases} 1;\text{if job } i \text{ comes before job } j \text{ on machine } k \\ 0;\text{otherwise} \end{cases}$$

$$Min\ C_{\max} \tag{1}$$

s.t.

$$C_{max} \geq s_{i_{last},i} + p_{i_{last},i}; \forall i_{last} \tag{2}$$

$$\sum_{k=1}^{m} z_{ku} = 1; u = 1, \ldots, m \tag{3}$$

$$\sum_{u=1}^{m} z_{ku} = 1; k = 1, \ldots, m \tag{4}$$

$$s_{li} - s_{ki} \geq p_{ki} + t_{uv} - M((1 - z_{ku}) + (1 - z_{lv})); \text{for all } (k,i) \rightarrow (l,i) \in A \text{ and } u, v = 1, \ldots, m \tag{5}$$

$$s_{kj} - s_{ki} \geq p_{ki} - M\left(1 - x_{kij}\right); \text{for all} (k,i) \text{ and } (k,j) \tag{6}$$

$$s_{ki} - s_{kj} \geq p_{kj} - Mx_{kij}; \text{for all } (k,i) \text{ and } (k,j) \tag{7}$$

$$z_{00} = 1 \tag{8}$$

$$s_{0i} = 0; i = 1, \ldots, n \tag{9}$$

$$s_{ki} \in \mathrm{N}^{+}; i = 1, \ldots, n; k = 1, \ldots, m \tag{10}$$

$$z_{ku} \in \{0, 1\}; k, u = 0, 1, \ldots, m \tag{11}$$

$$x_{kij} \in \{0, 1\}; \text{for all} (k,i), (k,j) \tag{12}$$

In this model, the objective function (1) minimizes the makespan. Constraint (2) shows that makespan is not less than the finish time of the last operation of each job. In this constraint, index i_{last} indicates the machine that processed the last operation of job i. Constraints (3) and (4) assure that each location is assigned to one machine and each machine is assigned to one location, respectively. If job i should be processed by machine k before being processed by machine l, the constraint $(k, i) \rightarrow (l, i)$ is proposed where symbol $\rightarrow$ represents the precedence constraints between two operations (k, i) and (l, i). In this case, the start time of job i on machine l should be greater or equal to the start time of that job on machine k plus its processing time on machine k and transportation time between machines k and l. If machines k and l are placed at locations u and v, respectively, we can show this restriction by constraint (5) in which M is a big positive integer. Also, constraint (5) prevents processing a job on two or more machines at a time. Since it is assumed that each machine can process at most one job at a time, when two different jobs require the same machine, one should precede the other. This restriction is shown in constraints (6) and (7). Constraint (8) ensures that the location of the warehouse is fixed, and constraint (9) indicates that start time of all jobs on machine zero (warehouse) is zero, implying the ready times of all jobs are zero. Constraint (10) shows that variable s_{ki} gets nonnegative integers, and constraints (11) and (12) represent that decision variables z_{ku} and x_{kij} are binary variables.

4 Solution representation and generation scheme

Each solution **S** of the CLSPJE is shown by a double vector **S** = (**s**, **q**) in which **s** = (s_{ki}); $\forall(k, i)$ where integer s_{ki} indicates the start times of operation (k, i) and **q** = $(q_1,\ldots, q_m)$ where $q_k \in \{1, 2,\ldots, m\}$ indicates the location of machine k. Our developed metaheuristic procedure is based on two algorithms, i.e., a constructive heuristic algorithm and an improving metaheuristic. Our constructive heuristic algorithm constructs solutions and our metaheuristic algorithm improves them.

Our constructive heuristic algorithm uses a solution representation to encode a solution and uses a solution generation scheme (SGS) to translate the solution representation to a solution **S** = (**s**, **q**). The SGS determines how a feasible solution is constructed by assigning the machines to the locations and the start times to the operations, whereby the locations and relative priorities are determined by the solution representation. A solution **S** of the CLSPJE is represented by a double list $\boldsymbol{\omega} = (\boldsymbol{\lambda}, \mathbf{q})$. The $\boldsymbol{\lambda}-$ list where $\boldsymbol{\lambda} = (\lambda_1,\ldots, \lambda_\pi)$ is a list of operations, a sequence of operations where λ_i is the operation number with ith priority for being scheduled using the SGS. Also, π belongs to $\{m, m + 1,\ldots, mn\}$ because it is assumed that there is at least one operation and at most n operations on each machine. The $\mathbf{q}-$ list where $\mathbf{q} = (q_1,\ldots, q_m)$ is a list of locations, where q_k represents the location assigned to machine k.

Since the JSP is a special version of the resource-constrained project scheduling problem (RCPSP) [27], activities and renewable resources in the RCPSP correspond to operations and machines in the JSP, respectively. Also, the dependencies among the operations can be shown by a network of operations in which each real operation has exactly one predecessor and one successor. Based on these similarities, we use ideas developed in the literature for the RCPSP to design more efficient solution representation for the CLSPJE. An operation list is known as precedence feasible if every operation is positioned after its predecessor, determined based on operations' order of jobs. We also enforce the topological order (TO) condition introduced by Valls et al. [28] in our operation list representation. To embed the TO condition in a given operation list, at first, operations are scheduled using a serial SGS (SSGS) [29] and then they are sequenced based upon nonincreasing order of their finish times. In other words, for all i and j, if $f_i(\mathbf{S}) > f_j(\mathbf{S})$, where $f_i(\mathbf{S})$ and $f_j(\mathbf{S})$ denote the finish times of operations i and j in solution $\mathbf{S}$, respectively, operation i comes before operation j in the topologically ordered operation list (TOOL). The advantage of this is that, whereas several operation lists can result in the same solution using a serial SGS (SSGS), each topological order corresponds to a unique solution, except in the case of identical operations finish times. Each operation list obtained using the TO condition is also a precedence feasible operation list for the reverse network in which all dependencies of the operation network are reversed. The TO condition was applied successfully with the concept of the reverse network in a scatter search metaheuristic in Ref. [30].

Each solution representation is translated to a real solution $\mathbf{S}$ using the SSGS. The SSGS is chosen because it generates active schedules and is robust against the precedence infeasible operation list [27]. In order to translate a solution representation $\boldsymbol{\omega} = (\boldsymbol{\lambda}, \mathbf{q})$ to a solution $\mathbf{S}$, the machines are assigned to the sites according to the given list $\mathbf{q}$. Then, a list scheduling approach is applied, i.e., operations are selected one by one and are scheduled as soon as possible. Selection of the operations is based on the priorities specified by $\boldsymbol{\lambda}-$list as well as the dependencies among operations. Each selected operation (k, i) can be started at time instant t if two conditions are satisfied. Firstly, finish time of its precedent operation plus transportation time between these two operations is not greater than t. Secondly, machine k is free from time instant t to $t + p_{ki}$.

5 The scatter search algorithm

The SS is an evolutionary method that creates new solutions by combining elite available solutions. For a general introduction to SS, we refer the reader to Ref. [31]. Figure 1 shows the main structure of our developed SS procedure.

In the first step, an initial population P containing $|P|$ solutions is generated. In the second step, the reference set (RefSet) including RefSet_1 and RefSet_2 is constructed. Solutions of RefSet_1 and RefSet_2 are called reference solutions. Next, NewSubsets is generated where each contains two reference solutions. Subsequently, two solutions of each subset are combined and new solutions are generated using a new combination method. The idea of the combination method is taken from the PR algorithm which explores a set of solutions on the path between the solutions in the selected subset solutions. The combination of each pair of solutions results in two new solutions, and the quality of these new obtained solutions may be improved by a local search with probability P_{ls}. Then, the TO condition is applied to the best solution so far and is transferred to the new population. In the next step, the dependencies in the operation network are reversed to perform a forward–backward improvement scheduling step [32]. More precisely, our developed procedure uses direct schedules, obtained from the operation network with original dependencies, to generate reverse children and vice versa. In other words, our procedure is a bipopulation (bpop) SS including direct and reverse populations. Steps 2–9 are repeated until the termination criterion, considered as a number of generated schedules (ns), is reached.

5.1 Construction of initial population P

An SS must be initialized with a starting population. The methods for creating an initial population $|P|$ are varied: feasible only, randomized, using heuristics, etc. For this problem, we choose to use randomized approach for the

Fig. 1 Scatter search procedure

```
1. Construct an initial population $P$ with size of $|P|$.
2. Build the reference set using the RefSet building method.
3. Generate Newsubsets with the subset generation method. Make $P=\varnothing$.
while (Newsubsets$\neq\varnothing$) do
    4. Select the next subset $\sigma$ in NewSubsets.
    5. Apply the solution combination method to $\sigma$ to obtain a number of new solutions.
    6. Select one of the new generated solutions in step 5 randomly, apply the local search
       over each selected solution with probability of $P_{ls}$ and add the new obtain solutions to
       the $P$.
    7. Delete $\sigma$ from subsets.
end while
8. Apply the TO-condition to the best solution so far and add it to the $P$.
9. Reverse activity network
10. If one the termination criterion is met, stop. Otherwise, go to 2.
```

following reasons. Firstly, randomized approach increases diversity of solutions quickly. Secondly, in terms of machine assignment, infeasible solutions are not created because the assumptions of the QAP are considered. Lastly, the SSGS is robust against precedence infeasible operation list. The size of $|P|$ is dependent to the size of RefSet which depends on the termination criterion (ns).

5.2 The local search

A local search procedure is applied over each generated solution **S** with probability p_{ls}. The general structure of this local search is constituted of a while loop that will be repeated at most $m/2$ times. At the first iteration of this procedure, locations of two machines, having the minimum distance, are exchanged while locations of other machines and the operation list $\boldsymbol{\lambda}$ remain unchanged. By applying the SSGS to the new generated $\boldsymbol{\omega}$, a new solution is obtained. If the makespan of this solution is not improved, the procedure is stopped; otherwise, the TO condition is applied on the newly obtained solution. The resulting solution is replaced with the corresponding unimproved solution in P. If any improvement is achieved in this iteration, the loop will be repeated by exchanging locations of two machines having the second minimum distance at the next iteration. This procedure is stopped whenever no more improvement is attained or the locations of two machines with the maximum distance have been exchanged.

In terms of schedule types, known as direct and reverse, all solutions of each population have the same schedule type. On the other hand, the TO condition converts schedules from direct to reverse or vice versa. Since the local search procedure may not be applied over all solutions of a population, modification is required. In order to preserve the schedule type, after applying the SSGS, operations are sequenced in nondecreasing order of their start times. This idea is a basis for construction of single population (spop) SS in which entire generated schedules are direct. We compare the performance of the bpop and spop versions of our developed SS in Section “5.3”.

5.3 Reference set building and subset generation methods

Unlike a traditional genetic algorithm that acts on the entire population, the SS maintains a smaller reference set of members separated from the larger population. The best b members (elite) of the population are copied to the RefSet. A reference set is a small set of solutions that contains not only the best solutions but also solutions to maintain diversity. The construction of high quality solutions, RefSet_1, starts with the selection of the solution in P with the lowest makespan. The representation of this solution is added to RefSet_1 as $\boldsymbol{\omega}_1$ and is deleted from P. The next best solution with representation $\boldsymbol{\omega}$ in P is chosen and added to RefSet_1 only if $D_{\min}(\boldsymbol{\omega})\geq t_1$, where $D_{\min}(\boldsymbol{\omega})$ is the minimum of the distances of solution representation $\boldsymbol{\omega}$ to the elements currently in RefSet, and t_1 is a threshold distance. The distance between two solution representations $\boldsymbol{\omega}^{\alpha}$ and $\boldsymbol{\omega}^{\beta}$ is calculated as Eq. (13).

$$D\left(\omega^{\alpha},\omega^{\beta}\right)=\frac{\sum_{i=1}^{nm}D\left(\lambda_i^{\alpha},\lambda_i^{\beta}\right)+\sum_{k=1}^{m}D\left(q_k^{\beta},q_k^{\beta}\right)}{nm+m} \tag{13}$$

where $D\left(\lambda_i^{\alpha},\lambda_i^{\beta}\right)=\begin{cases}1;\text{if } \lambda_i^{\alpha}\neq\lambda_i^{\beta}\\0;\text{otherwise}\end{cases}$ and $D\left(q_k^{\beta},q_k^{\beta}\right)=\begin{cases}1;\text{if } q_k^{\beta}\neq q_k^{\beta}\\0;\text{otherwise}\end{cases}$

This process is repeated until b_1 elements are selected for RefSet_1. To construct diverse solutions in RefSet_2, the same strategy is followed but with $t_2>t_1$ and with $D_{\min}(\boldsymbol{\omega})$ as the minimum distance to the elements in both RefSet_1 and RefSet_2. Therefore, both RefSet_1 and RefSet_2 contain diversified solutions, but with more emphasis on diversification in RefSet_2. When no qualified solution can be found in the population, RefSet is completed with randomly generated solutions based upon the method used for building the initial population. In this case, the minimum threshold distance condition is not checked for generated solutions.

After the RefSet construction, NewSubsets are generated. Each subset includes two solutions from RefSet_1 or one solution from RefSet_1, and the other one from RefSet_2. Therefore, since $|\text{RefSet}_1|=b_1$ and $|\text{RefSet}_2|=b_2$, then $|\text{NewSubsets}|=b_1(b_1-1)/2+b_1b_2$.

5.4 Solution combination method

Only solutions in the reference set are candidates to be used for combining to regenerate the new large population, unlike using the whole population in a genetic algorithm. Our developed combination method is based on the idea of the PR algorithm, referred to as PR combination method. In this approach, the solutions of each subset, generated from RefSet, are combined. In order to relatively measure the performance of the PR combination method, we also apply the two-point crossover method [33], usually used in the genetic algorithm, as the combination method in our SS procedure. This approach is referred to as the TC combination method.

5.4.1 The PR combination method

The PR has been suggested as an approach to integrate intensification and diversification strategies [34]. This approach generates new solutions by exploring trajectories connecting high-quality solutions. In the PR combination method, a movement is started from one of these solutions called initiating solution ($\boldsymbol{\omega}^{in} = (\boldsymbol{\lambda}^{in}, \mathbf{q}^{in})$), and a path is generate in the neighborhood space which leads toward another solution called guiding solution ($\boldsymbol{\omega}^{gu} = (\boldsymbol{\lambda}^{gu}, \mathbf{q}^{gu})$). Then, the role of initiating and the guiding solutions are exchanged to generate a new path in the opposite direction. In each direction, the path is constructed based on the moves that introduce attributes contained in the guiding solution. In each path, a set of solutions will be generated that one of them will be selected randomly as the child solution. After applying the TO condition to the child solution, it will be added to the new population.

In order to combine two arbitrary solutions $\boldsymbol{\omega}^{in}$ and $\boldsymbol{\omega}^{gu}$, $\boldsymbol{\lambda}^{in}$ is changed gradually based upon an iterative changing method including two steps. At the first step of the changing method, the first position i (from the left) in $\boldsymbol{\lambda}^{in}$ where $\boldsymbol{\lambda}^{in}(i) \neq \boldsymbol{\lambda}^{gu}(i)$ should be found, where for every $j < i$, $\boldsymbol{\lambda}^{in}(j) = \boldsymbol{\lambda}^{gu}(j)$. Then, the first position j in $\boldsymbol{\lambda}^{in}$, where $j > i$, such that $\boldsymbol{\lambda}^{in}(j) = \boldsymbol{\lambda}^{gu}(i)$ should be detected. Now, two elements $\boldsymbol{\lambda}^{in}(i)$ and $\boldsymbol{\lambda}^{in}(j)$ are exchanged. This process is repeat until we $\boldsymbol{\lambda}^{in}$ and $\boldsymbol{\lambda}^{gu}$ are the same ($\boldsymbol{\lambda}^{in} = \boldsymbol{\lambda}^{gu}$). Each change in $\boldsymbol{\lambda}^{in}$ generates a new $\boldsymbol{\lambda}-$ list, and all changes over $\boldsymbol{\lambda}^{in}$ result in a set of new lists of $\boldsymbol{\lambda}$. From this set, one list is selected randomly as a part of the child solution and shown by $\boldsymbol{\lambda}^{c}$. Similarly, the changing method is applied over $\mathbf{q}^{in}$ to reach $\mathbf{q}^{c}-$ list as the other part of the child solution. After that, the TO condition is applied to $\boldsymbol{\omega}^{c} = (\boldsymbol{\lambda}^{c}, \mathbf{q}^{c})$ and then its output is added to the new population. It should be considered that the generated $\boldsymbol{\lambda}^{c}$ may not be precedence feasible but it does not matter because the SSGS considers both the priorities and dependencies among operations concurrently.

5.4.2 The TC combination method

In the TC combination method, we deal with the parent solutions, $\boldsymbol{\omega}^{f} = (\boldsymbol{\lambda}^{f}, \mathbf{q}^{f})$ (father) and $\boldsymbol{\omega}^{m} = (\boldsymbol{\lambda}^{m}, \mathbf{q}^{m})$(mother), corresponding to the initiating and guiding solutions in the PR combination method. In this method, $\boldsymbol{\omega}^{f}$ and $\boldsymbol{\omega}^{m}$ are combined to generate two new solutions referred to as son solution ($\boldsymbol{\omega}^{s} = (\boldsymbol{\lambda}^{s}, \mathbf{q}^{s})$) and daughter solution ($\boldsymbol{\omega}^{d} = (\boldsymbol{\lambda}^{d}, \mathbf{q}^{d})$). For this purpose, the integer crossing points in lists $\boldsymbol{\lambda}$ and $\mathbf{q}$ are selected randomly. After this, the elements of $\boldsymbol{\lambda}^{s}$ located between crossing points are copied directly from $\boldsymbol{\lambda}^{f}$, and the rest are copied from $\boldsymbol{\lambda}^{m}$. By swapping the role of $\boldsymbol{\lambda}^{f}$ and $\boldsymbol{\lambda}^{m}$ and following the same rule, $\boldsymbol{\lambda}^{d}$ is generated. Similarly, $\mathbf{q}^{s}$ and $\mathbf{q}^{d}$ are generated from $\mathbf{q}^{f}$ and $\mathbf{q}^{m}$.

5.5 Modification of the SS algorithm for the JSP

Although we have designed our procedure for the CLSPJE, it can be modified to tackle the JSP. Since the CLSPJE is a generalization of the JSP, it is sufficient to consider fixed assignment for sites and machines with zero transportation times. Thus, the $\boldsymbol{\lambda}-$ list is enough for representation of a JSP solution. Also, we need to modify the formula (13) as the following where definition of $D\left(\lambda_i^{\alpha}, \lambda_i^{\beta}\right)$ is not changed:

$$D\left(\lambda^{\alpha}, \lambda^{\beta}\right) = \frac{\sum_{i=1}^{nm} D\left(\lambda_i^{\alpha}, \lambda_i^{\beta}\right)}{nm}. \tag{14}$$

Furthermore, a local search procedure is developed for the JSP as follows. This procedure gets a $\boldsymbol{\lambda}-$ list as an input and, for a direct schedule, it finds the operation having maximum distance with its predecessor in the list. Then, it puts this operation beside its predecessor and shifts all operations located between the selected operation and its predecessor to the right by one time unit. Next, the new obtained list ($\boldsymbol{\lambda}^{\text{new}}$) is evaluated by applying the SSGS. If the makespan is improved, the procedure is repeated; otherwise it is stopped. At the best case, the improvement is repeated until each operation is located besides its predecessor in the $\boldsymbol{\lambda}-$ list. For a reverse schedule, the successor of each operation is considered instead of its predecessor in the above-mentioned procedure.

6 Computational results

6.1 Benchmark problem sets

The procedure is coded in Visual C++ 6 and all computational experiments are performed on a personal computer Pentium IV 3 GHz processor with single core CPU and with 1,024 MB of internal memory. Since the CLSPJE is proposed for the first time and there was no test set in the literature for that, we consider 43 benchmark instances of the JSP belonging to two classical sets known as LA from Lawrence [35] and FT from Fisher and Thompson [36]. Each benchmark instance of the JSP is modified for the CLSPJE by adding the distance between each pair of sites, a stochastic variable with discrete uniform distribution U [1,100]. The problem size varies between 6 and 30 jobs and between 5 and 15 machines. It is well-known that some problems are more difficult than others. Usually, JSP problems become harder to solve as the problem size increases or the number of jobs gets closer to the number of machines. The modified benchmark instances of Lawrence are referred to as LAC, and the modified benchmark instances of Fisher and Thompson are referred to as FTC.

6.2 Parameters setting

Using a full factorial design, the values of parameters are adjusted for the CLSPJE and the JSP. We consider five levels for each parameter shown in Table 1.

Table 1 Levels of parameters

Parameters	Levels				
t_1	0.3	0.4	0.5	0.6	0.7
t_2	0.5	0.6	0.7	0.8	0.9
p_{ls}	0.00	0.01	0.02	0.03	0.04
b	$(ns)^{0.2}$	$(ns)^{0.3}$	$(ns)^{0.4}$	$(ns)^{0.5}$	$(ns)^{0.6}$
b_1	$0.3\times b$	$0.4\times b$	$0.5\times b$	$0.6\times b$	$0.7\times b$

In order to evaluate the impact of the termination criterion on the quality of solutions, three values 5,000, 50,000 and 500,000 schedules are considered for parameter ns. Henceforth, the SS procedure with ns = 5,000, 50,000 and 500,000 schedules is referred to as SS_L, SS_M and SS_H, respectively. Since there are numerous (5^5=3,125) different designs, we run only SS_L for adjusting the parameters. The best values of parameters are shown in Table 2 for the CLSPJE. For the JSP, the identical results are obtained for best values of parameters except the parameter t_2 where its best value is 0.7.

6.3 Comparative computational results

6.3.1 Impact of the scatter search components

Since the CLSPJE is proposed for the first time, we could not compare the efficiency of our algorithm with any previous work. Thus, in this section, we analyze the effect of different components of the SS. For each component of algorithm, superiority of a scenario against other scenarios is investigated. These components include the type of population, the local search and the combination method.

The results shown in Table 3 are obtained based on the default SS procedure including the setting mentioned above and the PR combination method. In this table, the first column indicates the problem name, the second column shows the number of jobs, the third column displays the number of machines and the forth column represents the best-known solutions (BKS) obtained based upon the all runs. The next nine columns, categorized to three sets, report three items for SS_L, SS_M and SS_H including the best found solutions

Table 2 Best values of parameters for the CLSPJE

Parameters	t_1	t_2	p_{ls}	b	b_1
Best value	0.6	0.8	0.03	$(ns)^{0.3}$	$0.4\times b$

(BFS), the corresponding percentage deviation (PD) relative to the BKS values and the time consumed to reach the BFS (in seconds). All test problems of the CLSPJE and the best found solutions can be downloaded from https://sites.google.com/site/clspje.

Impact of the population type In this section, the results of two versions of our developed SS procedure, bpop (SS^{bpop}) and spop (SS^{spop}) are compared. The difference between these two types of procedures emanates from the rule of TO condition where in the SS^{bpop}, the TO condition is applied based on the finish times of operations while in the SS^{spop}, the TO condition is applied based on the start times of operations. Table 4 is the summary of Table 3 related to the SS^{bpop}, and Table 5 has a similar structure but obtained from running of the SS^{spop} procedure. In these tables, we have grouped the LAC benchmark instances to eight groups based on the number of jobs and the number of machines. Also, for each value of ns, we have reported the number of BKS founded by the procedure (#BKS), average PD (APD) and the time consumed to reach the BFS (in seconds).

Overall, the results reveal that the SS^{bpop} has better performance than SS^{spop}. The SS_L^{bpop}, SS_M^{bpop} and SS_H^{bpop} procedures have found 8, 15 and 35 out of the 43 best-known solutions, respectively, while SS_L^{spop}, SS_M^{spop} and SS_H^{spop} procedures have found 4, 8 and 12 out of the 43 best-known solutions, respectively. In addition, the APD of the SS^{bpop} approach is noticeably smaller than the SS^{spop} approach, while their average CPU times (to reach the BSF) are almost equal. Since the SS^{spop} has better performance than the SS^{spop}, the results of the next sections are obtained by applying the SS^{bpop} approach.

Impact of the local search In order to evaluate the impact of the local search, we compare the results of the SS procedure with and without local search, i.e., P_{ls}=0.03 and P_{ls}=0. The results for the case that the local search is incorporated were shown in Table 4. Also, when the local search procedure is removed from the algorithm, we obtained the results presented in Table 6. For all SS_L, SS_M and SS_H, the #BKS and the APD of the SS procedure with local search are better than the SS procedure without local search, while their CPU times do not have evident difference. For the results of the other sections, we consider the SS with the local search.

Impact of the combination method The performance of the PR and the TC combination methods can be evaluated based on the data reported in Table 7 and comparing them with Table 4. Table 7 shows the results of running the SS procedure with the TC combination method. Obviously, it can be

Table 3 Detailed results of the default scatter search procedure

Problem				SS_L			SS_M			SS_H		
Name	*n*	*m*	BKS	BSF	PD	Time	BSF	PD	Time	BSF	PD	Time
LAC1	10	5	773	779	0.776	0.2	773	0.000	0.9	773	0.000	1.8
LAC2	10	5	750	751	0.133	0.2	751	0.133	0.8	750	0.000	1.9
LAC3	10	5	781	792	1.408	0.4	781	0.000	0.7	781	0.000	4.4
LAC4	10	5	741	741	0.000	0.2	741	0.000	0.9	741	0.000	5.4
LAC5	10	5	635	643	1.260	0.3	640	0.787	0.5	635	0.000	3.7
LAC6	15	5	936	936	0.000	0.8	936	0.000	1.2	936	0.000	6.0
LAC7	15	5	966	980	1.449	0.7	973	0.725	1.7	966	0.000	8.6
LAC8	15	5	891	929	4.265	0.9	899	0.898	1.3	891	0.000	9.5
LAC9	15	5	978	981	0.307	1.0	978	0.000	2.1	978	0.000	7.6
LAC10	15	5	968	968	0.000	0.2	968	0.000	1.4	968	0.000	3.2
LAC11	20	5	1,241	1,241	0.000	0.6	1,241	0.000	1.0	1,241	0.000	5.5
LAC12	20	5	1,049	1,049	0.000	0.7	1,049	0.000	1.8	1,049	0.000	9.4
LAC13	20	5	1,151	1,151	0.000	0.7	1,151	0.000	1.9	1,151	0.000	8.3
LAC14	20	5	1,299	1,299	0.000	0.2	1,299	0.000	0.6	1,299	0.000	1.2
LAC15	20	5	1,224	1,262	3.105	1.5	1,237	1.062	3.6	1,224	0.000	21.8
LAC16	10	10	1,241	1,256	1.209	1.1	1,243	0.161	7.7	1,241	0.000	28.1
LAC17	10	10	1,074	1,091	1.583	1.0	1,074	0.000	3.4	1,074	0.000	33.7
LAC18	10	10	1,073	1,113	3.728	0.4	1,081	0.745	5.5	1,073	0.000	34.8
LAC19	10	10	1,116	1,139	2.061	1.3	1,125	0.806	5.3	1,119	0.269	27.7
LAC20	10	10	1,115	1,155	3.587	1.4	1,128	1.166	7.2	1,115	0.000	27.4
LAC21	15	10	1,317	1,385	5.163	2.7	1,353	2.733	13.1	1,317	0.000	46.3
LAC22	15	10	1,181	1,221	3.387	2.0	1,189	0.677	15.2	1,193	1.016	43.6
LAC23	15	10	1,250	1,285	2.800	2.1	1,253	0.240	10.8	1,250	0.000	39.3
LAC24	15	10	1,243	1,303	4.827	1.3	1,263	1.609	16.5	1,243	0.000	45.8
LAC25	15	10	1,230	1,238	0.650	2.6	1,247	1.382	22.5	1,230	0.000	52.4
LAC26	20	10	1,427	1,472	3.153	3.7	1,427	0.000	27.1	1,431	0.280	89.7
LAC27	20	10	1,475	1,533	3.932	3.4	1,514	2.644	24.1	1,475	0.000	101.7
LAC28	20	10	1,445	1,505	4.152	3.9	1,453	0.554	31.6	1,445	0.000	110.6
LAC29	20	10	1,407	1,472	4.620	4.0	1,435	1.990	36.4	1,423	1.137	94.7
LAC30	20	10	1,548	1,571	1.486	3.2	1,556	0.517	19.2	1,548	0.000	103
LAC31	30	10	1,808	1,871	3.485	5.6	1,822	0.774	25.6	1,808	0.000	117.6
LAC32	30	10	1,890	1,946	2.963	6.2	1,890	0.000	38.9	1,897	0.370	125.3
LAC33	30	10	1,749	1,814	3.716	7.1	1,770	1.201	43.2	1,749	0.000	138.9
LAC34	30	10	1,776	1,842	3.716	7.2	1,801	1.408	47.6	1,776	0.000	143.4
LAC35	30	10	1,933	1,989	2.897	6.5	1,942	0.465	38.8	1,933	0.000	154.6
LAC36	15	15	1,667	1,736	4.139	2.3	1,712	2.699	46.7	1,679	0.720	204.1
LAC37	15	15	1,777	1,858	4.558	2.8	1,819	2.363	31.5	1,799	1.238	227.6
LAC38	15	15	1,692	1,740	2.837	4.6	1,703	0.650	27.8	1,692	0.000	229
LAC39	15	15	1,665	1,719	3.243	3.8	1,679	0.841	16.2	1,674	0.540	231.1
LAC40	15	15	1,690	1,735	2.663	4.0	1,700	0.592	25.6	1,690	0.000	256.5
FTC1	6	6	368	368	0.000	0.0	368	0.000	0.0	368	0.000	0.1
FTC2	10	10	1,187	1,227	3.370	1.1	1,216	2.443	17.2	1,187	0.000	73.2
FTC3	10	5	1,298	1,385	6.703	1.7	1,298	0.000	15.3	1,298	0.000	17.2

concluded that the TC combination method is dominated by the PR combination method especially in terms of the #BKS. Of course, the CPU time for the TC in all groups is smaller than the PR but this happens due to the quick convergence of the procedure. Also, in terms of the total consumed CPU time, not reported in the Tables 4 and 7, the

Table 4 Summary results of the SS^{bpop}

Problem			SS_L^{bpop}			SS_M^{bpop}			SS_H^{bpop}		
Name	*n*	*m*	#BKS	APD	Time	#BKS	APD	Time	#BKS	APD	Time
LAC1–5	10	5	1	0.716	0.3	3	0.184	0.8	5	0.000	3.4
LAC6–10	15	5	2	1.204	0.7	3	0.324	1.5	5	0.000	7.0
LAC11–15	20	5	4	0.621	0.7	4	0.212	1.8	5	0.000	9.2
LAC16–20	10	10	0	2.433	1.0	1	0.576	5.8	4	0.054	30.3
LAC21–25	15	10	0	3.365	2.1	0	1.328	15.6	4	0.203	45.5
LAC26–30	20	10	0	3.469	3.6	1	1.141	27.7	3	0.283	99.9
LAC31–35	30	10	0	3.355	6.5	1	0.770	38.8	4	0.074	135.9
LAC36–40	15	15	0	3.488	3.5	0	1.429	29.6	2	0.500	229.7
All LAC			7	2.331	2.3	13	0.745	15.2	32	0.139	70.1
FTC1	6	6	1	0.000	0.0	1	0.000	0.1	1	0.000	0.1
FTC2	10	10	0	3.370	1.1	0	2.443	17.1	1	0.000	73.2
FTC3	10	5	0	6.703	1.7	1	0.000	15.3	1	0.000	17.2
All FTC			1	3.357	0.9	3	0.814	10.8	3	0.000	30.2
All			8	2.403	2.2	15	0.750	14.9	35	0.130	67.3
Best				0.000	0.0		0.000	0.0		0.000	0.1
Worst				6.703	7.2		2.733	47.6		1.238	256.5

TC is a bit faster than the PR because in the PR method, we generate a set of solutions in each path and select one of them, while in the TC method, each generated solution is selected. It should be noticed that the computational results reported in other sections are obtained by applying the PR combination method.

6.3.2 Comparative results for the JSP

Although our developed SS procedure is devoted to the CLSPJE, we modified and applied it to the JSP to show its relative efficiency. Table 8 indicates the results of our developed SS procedure based on 40 instances of Lawrence [35].

Table 5 Summary results of the SS^{spop}

Problem			SS_H^{spop}			SS_M^{spop}			SS_L^{spop}		
Name	*n*	*m*	#BKS	APD	Time	#BKS	APD	Time	#BKS	APD	Time
LAC1–5	10	5	0	2.731	0.4	1	1.600	1.0	1	0.600	4.8
LAC6–10	15	5	1	2.752	0.8	2	1.567	1.9	3	0.640	8.2
LAC11–15	20	5	3	1.892	0.8	4	0.882	2.2	4	0.376	9.9
LAC16–20	10	10	0	5.313	1.1	0	2.442	6.7	2	0.573	34.4
LAC21–25	15	10	0	7.241	2.0	0	3.635	16.1	0	2.394	45.8
LAC26–30	20	10	0	8.973	3.0	0	5.308	26.4	0	2.679	105.9
LAC31–35	30	10	0	9.233	6.7	0	4.979	35.3	0	2.880	134.9
LAC36–40	15	15	0	8.035	4.1	0	4.948	34.6	0	2.426	231.2
All LAC			4	5.771	2.4	7	3.170	15.5	10	1.571	71.9
FTC1	6	6	0	0.543	0.2	1	0.000	0.3	1	0.000	0.2
FTC2	10	10	0	6.318	1.2	0	2.275	13.5	1	0.000	77.3
FTC3	10	5	0	9.014	1.4	0	4.700	15.7	0	1.156	12.7
All FTC			0	5.292	0.9	1	2.325	9.8	2	0.385	30.1
All			4	5.738	2.3	8	3.111	15.1	12	1.488	69.0
Best				0.000	0.2		0.000	0.3		0.000	0.2
Worst				12.225	7.4		7.605	48.9		5.544	251.5

Table 6 Summary results of the scatter search without local search

Problem			SS_H			SS_M			SS_L		
Name	*n*	*m*	#BKS	APD	Time	#BKS	APD	Time	#BKS	APD	Time
LAC1–5	10	5	0	1.702	0.4	2	1.006	0.9	3	0.232	4.5
LAC6–10	15	5	1	1.690	0.8	3	0.583	2.1	3	0.066	8.5
LAC11–15	20	5	3	0.588	1.0	4	0.229	2.0	5	0.000	10.6
LAC16–20	10	10	0	3.788	1.4	2	1.314	6.1	3	0.124	30.1
LAC21–25	15	10	0	4.651	2.1	0	2.075	14.1	1	1.028	44.3
LAC26–30	20	10	0	6.029	3.7	0	2.134	25.8	1	0.452	83.7
LAC31–35	30	10	0	4.322	5.8	0	2.083	37.8	0	0.792	133.9
LAC36–40	15	15	0	4.790	4.2	0	2.268	33.7	4	0.319	218.4
All LAC			4	3.445	2.4	11	1.462	15.3	20	0.376	66.7
FTC1	6	6	1	0.000	0.1	1	0.000	0.1	1	0.000	0.2
FTC2	10	10	0	4.212	1.3	0	3.033	16.1	0	1.516	91.4
FTC3	10	5	0	6.163	1.6	0	5.008	13.3	0	1.233	68.7
All FTC			1	3.459	1.0	1	2.680	9.8	1	0.916	53.4
All			5	3.446	2.3	12	1.547	14.9	21	0.414	65.8
Best				0.000	0.1		0.000	0.1		0.000	0.2
Worst				7.249	7.0		5.008	48.9		2.885	236.7

To our knowledge, the four best metaheuristic developed for the JSP are tabu search algorithm of Grabowski and Wodecki (TSGW) [12], tabu search approach of Nowichi and Smutnicki (TSAB) [13], hybrid TS/SA algorithm (TSSA) of Zhang et al. [15] and the algorithm of Pardalos and Shylo [21] (GES1), designed based on the global equilibrium search techniques. Table 9 shows the available results of these algorithms on the test set of Lawrence.

The inclusion of the our SS algorithm in the analysis gives an average relative deviation from the best-known solutions of 0.68% for the SS_L approach, 0.40% for the SS_M approach and 0.18% for the SS_H approach. Also, the average relative deviation from the best-known solutions is 0.05% for the TSGW approach, 0.07% for the TSAB approach, 0.05% for the TSSA approach and 0.010% for the GES1 approach.

Table 7 Summary results of the scatter search with the TC combination method

Problem			SS_H			SS_M			SS_L		
Name	*n*	*m*	#BKS	APD	Time	#BKS	APD	Time	#BKS	APD	Time
LAC1–5	10	5	0	6.847	0.3	0	5.613	0.7	0	3.046	4.3
LAC6–10	15	5	0	9.652	0.5	0	5.854	1.8	0	3.951	8.8
LAC11–15	20	5	0	7.138	0.5	0	6.174	2.2	0	3.980	17.2
LAC16–20	10	10	0	15.375	0.9	0	8.367	6.9	0	6.698	26.1
LAC21–25	15	10	0	20.984	0.7	0	16.219	11.2	0	10.459	39.3
LAC26–30	20	10	0	20.484	0.7	0	17.604	19.6	0	15.886	71.2
LAC31–35	30	10	0	18.682	1.9	0	16.380	25.5	0	16.501	45.1
LAC36–40	15	15	0	16.359	1.8	0	13.753	23.1	0	9.877	196.1
All LAC			0	14.440	0.9	0	11.245	11.4	0	8.800	51.0
FTC1	6	6	1	0.000	0.0	1	0.000	0.0	1	0.000	0.1
FTC2	10	10	0	18.703	0.7	0	10.194	6.7	0	9.183	89.7
FTC3	10	5	0	13.867	0.4	0	11.787	8.0	0	6.548	15.2
All FTC			1	10.857	0.37	1	7.327	4.9	1	5.244	35.0
All			1	14.190	0.9	1	10.972	10.9	1	8.552	49.9
Best				0.000	0.0		0.000	0.0		0.000	0.1
Worst				24.176	3.7		19.629	39.1		18.363	241.6

Table 8 Summary results of the scatter search for the JSP

Problem			SS_L		SS_M		SS_H	
Name	*n*	*M*	APD	Time	APD	Time	APD	Time
LA1–5	10	5	0.000	0.3	0.000	0.4	0.000	0.7
LA6–10	15	5	0.000	0.1	0.000	0.3	0.000	0.6
LA11–15	20	5	0.000	0.3	0.000	0.5	0.000	0.9
LA16–20	10	10	0.132	0.9	0.000	3.5	0.000	8.0
LA21–25	15	10	1.676	1.9	0.653	11.0	0.316	31.8
LA26–30	20	10	1.855	3.4	1.179	21.3	0.444	102.6
LA31–35	30	10	0.000	3.8	0.000	6.6	0.000	12.0
LA36–40	15	15	1.816	3.7	1.398	29.7	0.683	119.9
Average			0.685	1.8	0.404	9.2	0.180	34.6

The average computation times for the TSGW, the TSAB, the TSSA and the GES1 are almost 1 s (on a 333-MHz CPU), 82.5 s (on a 386DX CPU), 13 s (on a 330 Sparc CPU) and 17.7 s (on a 2.8-GHz CPU), respectively.

In terms of solution quality and computational time, we conclude that the GES1 is the best available metaheuristic for the JSP but its average computational time is relatively high for larger test instances. After that, the TSGW has the best solution quality. Although the comparison of the results reveals that four other efficient approaches are better than SS_H in terms of solution quality, but the difference is not noticeable and our developed SS procedure is efficient as well. On the other hand, it should be noticed that these four approaches are the best ones in the literature and our developed metaheuristic is better than almost all other approaches presented in a recently published paper [18].

In order to compare our algorithm with very efficient algorithms of Refs. [14] and [22], we tested our developed SS on the test set of Taillard [37] for test instances Ta01–Ta10. We consider $RE[i] = 100\%\left(C_{Max}^{[i]} - C_{Max}^{*[i]}\right)/C_{Max}^{*[i]}$ as the comparison criterion where $C_{Max}^{[i]}$ is the best makespan value obtained by the SS for the *i*th test instance, and $C_{Max}^{*[i]}$ is the optimal makespan value for the *i*th problem. For the test instance Ta01–Ta10, the algorithm presented in Refs. [20] and [14] provide an average of RE[*i*] equal to 0.08% and 0.11%, respectively. Also, the average time to the best solution is equal to 24.6 s (on a 2.8-GHz CPU) and 26 s (on a 900-MHz CPU) for the results presented in Refs. [22] and [14], respectively. For a fair comparison, we changed our termination criterion to a time limit equal to 25 s. In this case, our algorithm provides an average of 0.53%.

7 Conclusion

In this study, we proposed the concurrent layout and scheduling problem in a job shop environment as a generalization of the job shop scheduling problem in which the locations of machines are not predetermined. It is assumed that each machine can be assigned to each possible site, and the transportation times between each pair of machines are only dependent to their distances. The problem is formulated as an integer programming model and is solved using a scatter search metaheuristic. We extended our procedure to the classic job shop scheduling problem, and the computational results reveal the efficiency of our developed procedure.

As a future research opportunity, we suggest to consider open shop scheduling problems while the locations of machines are not fixed. If the QAP assumption in which each machine can be placed in each location is removed, the problem will be more interesting. As another suggestion, this problem can be solved by other single objective functions or multiobjective functions. In addition, developing other efficient metaheuristic or even exact algorithms for the proposed problem can be an interesting research topic.

Table 9 Summary results of other metaheuristics for the JSP

Problem			TSGW		TSAB		TSSA		GES1	
Name	*n*	*m*	APD	Time	APD	Time	APD	Time	APD	Time
LA1–5	10	5	0.000	0.0	0.000	3.8	0.000	0.000	0.000	0.000
LA6–10	15	5	0.000	0.0	0.000	0.0	–	–	0.000	0.000
LA11–15	20	5	0.000	0.0	0.000	0.0	–	–	0.000	0.000
LA16–20	10	10	0.000	1.3	0.020	68.6	0.000	0.200	0.000	0.000
LA21–25	15	10	0.042	2.2	0.105	74.0	0.030	13.600	0.000	17.2
LA26–30	20	10	0.052	2.1	0.160	136.4	0.020	15.200	0.020	15.4
LA31–35	30	10	0.000	0.0	0.000	1.6	–	–	0.000	0.000
LA36–40	15	15	0.283	2.6	0.280	375.6	0.190	36.100	0.070	109.2
Average			0.047	1.0	0.071	82.5	0.048	13.0	0.011	17.7

References

1. Garey EL, Johnson DS, Sethi R (1976) The complexity of flowshop and job-shop scheduling. Math Oper Res 1:117–129
2. Hahn P, Grant T, Hall N (1998) A branch-and-bound algorithm for the quadratic assignment problem based on the Hungarian method. Eur J Oper Res 108:629–640
3. Erdogan G, Tansel B (2007) A branch-and-cut algorithm for quadratic assignment problems based on linearizations. Comput Oper Res 34:1085–1106
4. James T, Rego C, Glover F (2009) A cooperative parallel tabu search algorithm for the quadratic assignment problem. Eur J Oper Res 195:810–826
5. Taillard E (1991) Robust taboo search for the quadratic assignment problem. Parallel Comput 17(4–5):443–455
6. Ahuja RK, Orlin JB, Tiwari A (2000) A greedy genetic algorithm for the quadratic assignment problem. Comput Oper Res 27:917–934
7. Misevicius A (2004) An improved hybrid genetic algorithm: new results for the quadratic assignment problem. Knowl Base Syst 17:65–73
8. Stutzle T (2006) Iterated local search for the quadratic assignment problem. Eur J Oper Res 174:1519–1539
9. Hasegawa M, Ikeguchi T, Aihara K, Itoh K (2002) A novel chaotic search for quadratic assignment problems. Eur J Oper Res 139:543–556
10. Carlier J, Pinson E (1989) An algorithm for solving the job-shop problem. Manag Sci 35:164–176
11. Brucker P, Jurisch B, Sievers B (1994) A branch-and-bound algorithm for the job-shop scheduling problem. Discret Appl Math 49 (1–3):107–127
12. Grabowski J, Wodecki M (2005) A very fast tabu search algorithm for job shop problem. In: Rego C, Alidaee B (eds) Metaheuristic optimization via memory and evolution: tabu search and scatter search. Kluwer, Boston, pp 191–211
13. Nowicki E, Smutnicki C (1996) A fast taboo search algorithm for the job shop problem. Manag Sci 42(6):797–813
14. Nowicki E, Smutnicki C (2005) An advanced tabu search algorithm for the job shop problem. J Sched 8(2):145–159
15. Zhang CY, Li PG, Rao YQ, Guan ZL (2008) A very fast TS/SA algorithm for the job shop scheduling problem. Comput Oper Res 35:282–294
16. Van Laarhoven PJM, Aarts EHL, Lenstra JK (1992) Job shop scheduling by simulated annealing. Oper Res 40(1):113–125
17. Nasiri MM, Kianfar F (2011) A hybrid scatter search for the partial job shop. Int J Adv Manuf Technol 52(9):1031–1038
18. Rego C, Duarte R (2009) A filter-and-fan approach to the job shop scheduling problem. Eur J Oper Res 194:650–662
19. Udomsakdigool A, Khachitvichyanukul V (2011) Ant colony algorithm for multi-criteria job shop scheduling to minimize makespan, mean flow time and mean tardiness. Int J Manag Sci Eng Manag 6(2):117–123
20. Pratchayaborirak T, Kachitvichyanukul V (2011) A two-stage PSO algorithm for job shop scheduling problem. Int J Manag Sci Eng Manag 6(2):84–93
21. Pardalos P, Shylo O (2006) An algorithm for the job shop scheduling problem based on global equilibrium search techniques. Comput Manag Sci 3:331–348
22. Pardalos P, Shylo O, Vazacopoulos A (2010) Solving job shop scheduling problems utilizing the properties of backbone and “big valley”. Comput Optim Appl 47:61–76
23. Hurink J, Knust S (2005) Tabu search algorithms for job-shop problems with a single transport robot. Eur J Oper Res 162:99–111
24. Li B, Zhao ZY, Li G (2005) A dynamic scheduling method for spatial layout planning. Proceeding of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou
25. Wu X, Chu CH, Wang Y, Yue D (2007) Genetic algorithms for integrated cell formation with machine layout and scheduling. Comput Ind Eng 53:277–289
26. Pinedo ML (2008) Scheduling: theory, algorithms and systems. Springer Science
27. Sprecher A, Kolisch R, Drexl A (1995) Semi-active, active and non-delay schedules for the resource-constrained project scheduling problem. Eur J Oper Res 82(1):94–102
28. Valls V, Quintanilla MS, Ballestin F (2003) Resource-constrained project scheduling: a critical reordering heuristic. Eur J Oper Res 165(2):375–386
29. Kolisch R (1996) Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. Eur J Oper Res 90(2):320–333
30. Ranjbar M, De Reyck B, Kianfar F (2009) A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling. Eur J Oper Res 193:35–48
31. Laguna M, Marti R (2003) Scatter search—methodology and implementations in C. Kluwer, Boston
32. Li KY, Willis RJ (1992) An iterative scheduling technique for resource-constrained project scheduling. Eur J Oper Res 56:370–379
33. Sivanandam SN, Deepa SN (2009) Introduction to genetic algorithms. Springer
34. Glover F, Laguna M, Marti R (2000) Fundamentals of scatter search and path relinking. Control Cybern 39:653–684
35. Lawrence S (1984) Supplement to resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh
36. Fisher H, Thompson GL (1963) Probabilistic learning combinations of local job shop scheduling rules. In: Muth JF, Thompson GL (eds) Industrial scheduling. Prentice-Hall, Englewood Cliffs, pp 225–251
37. Taillard ED (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64(2):278–285