Research Article

International Journal of Advanced Computer Research, Vol 8(37)
ISSN (Print): 2249-7277 ISSN (Online): 2277-7970
http://dx.doi.org/10.19101/IJACR.2018.835001

A metaheuristic for solving flowshop problem

Peter Bamidele Shola[1*] and Asaju La'aro Bolaji[2]
Lecturer, Department of Computer Science, University of Ilorin, Nigeria[1]
Lecturer, Department of Computer Science, Federal University Wukari, Taraba State, Nigeria[2]

Received: 11-September-2017; Revised: 26-April-2018; Accepted: 15-May-2018

Abstract
Discrete optimization is a class of computational expensive problems that are of practical interest and consequently have attracted the attention of many researchers over the years. Yet no single method has been found that could solve all instances of the problem. The no free launch theorem that confirms that no such general method (that can solve all the instances) could be found, has limited research activities in developing method for a specific class of instances of the problem. In this paper an algorithm for solving discrete optimization is presented. The algorithm is obtained from a hybrid continuous optimization algorithm using a technique devised by Clerc for particle swarm optimization (PSO). In the method, the addition, subtraction and multiplication operators are redefined to support discrete domain. The effectiveness of the algorithm was investigated on the flowshop problem using the makespan as the performance measure and the Taillard benchmark problem instances as the dataset. The result of the investigation is presented in this paper and compared with those from some existing algorithms, including genetic algorithm (GA), ant colony optimization (ACO), simulated annealing (SA), firefly and cockroach algorithms. Based on the experimental results, the algorithm is proposed as a competitive or a viable alternative for solving flowshop problems and possibly discrete optimization problems in general.

Keywords
Flowshop, Combinatorial, Optimization, Metaheuristics.

1.Introduction
Many real-world optimization problems involve making discrete choices among a finite or countable number of options. Such problems, (collectively called the discrete optimization problems (DOPs)) arise in planning, budgeting, games and scheduling (time tabling, staff-roster, routing). Precisely, given a discrete finite search space S, a set of constraints and an objective function f :S−>R (where R is a set of real numbers), the discrete optimization is the problem of finding a point, $x^{\times}$ in S satisfying all the constraints with $f(x^{\times}) = \min_{x \epsilon S} f(x)$or $f(x^{\times}) = \max_{x \epsilon S} f(x)$.

Many methods have been developed for solving DOPs but each with its own limitation. This article presents an adaptation of the hybrid continuous optimization technique in [1] to solving DOPs and investigates its effectiveness of the flow shop problem (FSP). The FSP is a scheduling problem of finding the best order to process a set of jobs on a given number of machines (or processors) with respect to a performance measure [2].

There are many types of FSP [3]. We consider in this paper, the permutation flowshop problem (PFSP) with the following assumptions.

- Each machine processes one job at a time and only once.
- Each job requires processing on each machine (to complete).
- The machines process the jobs in the same order.
- The processing time of each job on each machine is known in advance and is assumed to include transportation and setup times.
- A store of unlimited capacity is available between successive machines to hold a job waiting for the next machine.
- All the jobs and machines are available at the start of processing.
- Each machine is always available when not on a job.
- Jobs are independent of each other (i.e. no precedence demand on the job scheduling).
- No preemption: A machine finishes a job before taking another.

*Author for correspondence

Many performance measures can be identified, for FSP. These (measures) include the makespan, total tardiness, mean flow time, maximum lateness and the machine setup cost. We adopt the makespan in this study. The makespan is the maximum completion time of the last job in a job sequence.

Given a set of jobs $J = \{J_1, J_2, \ldots\ldots, J_n\}$ for processing on m machines and a processing order, $\pi = J_{p_1}, J_{p_2}, \ldots\ldots, J_{p_n}$ (where $p_1, p_2, \ldots\ldots, p_n$ is just a permutation of integers 1,2,….n) then the completion time, $C(J_{p_i}, k)$, of job J_{p_i} on machine k can be computed from the recurrence relations:

$C(J_{p_1}, 1) = t(J_{p_1}, 1)$

$C(J_{p_1}, k) = C(J_{p_1}, k-1) + t(J_{p_1}, k)\ k = 1,2,..m$

$C(J_{p_i}, 1) = C(J_{p_{i-1}}, 1) + t(J_{p_i}, 1)\ i = 1,2,..n$

$C(J_{p_i}, k) = \max(C(J_{p_i}, k-1) + C(J_{p_{i-1}}, k)) + t(J_{p_i}, k) \quad i = 1,2,..n, \quad k = 1,2,..m$

Where $t(J_{p_i}, k)$ is the processing time of job J_{p_i} on machine k. The makespan, $MkSpan(\pi)$, for π is $C(J_{p_n}, m)$.

If Π is the set of all permutations of the jobs in J then the objective of PFSP is to find a permutation π^* in Π such that:

$$C(\pi^*) = \min_{\pi \epsilon \Pi} MkSpan(\pi)$$

The FSP (for the number of machines greater than 2) is NP-hard [4] and so the exhaustive enumeration technique, where every possible solution is examined to find the optimal one, is impracticable. A FSP with n jobs and m machines has $(n!)^m$ job arrangements. The search space could however be reduced to $(n!)^{m-1}$ due to the dominant properties of flowshop problems. It states that the optimal schedule could be found among those schedules that have the same job sequence on machine 1 and 2 and the same job sequence on the last two machines, (i.e. machines m-1 and m). The search space is further reduced to $(n!)^{m-2}$ where the performance measure is the makespan. The PFSP however, has n! job arrangements, but this has not changed its time complexity. It is still NP-hard.

Some exact methods have been applied to solve FSP. The branch and bound technique were used in [5, 6] to obtain the exact solutions to some FSPs. In the method the known (lower or upper) bound on the solution is used to guide the search. The exploration of a solution path is discontinued at the point the bound is exceeded. The time the method takes to arrive at a solution makes it unsuitable for FSPs with large number of machines and/or jobs. The dynamic programming technique applied in [7, 8] is computationally expensive.

Many approximate methods have been devised for solving PFSP. Unlike the exact method, the approximate methods, could find only "good" or near optimal solutions but in a relatively short time. Most approximate methods are heuristic or metaheuristic based and so devoid of mathematical proof. Johnson [9] developed a heuristic that finds the optimal solution for PFSP with two machines in O (nlogn) time. Another heuristic approach has been presented in [10, 11] that solved the 2-machines PFSP. Campbell et al. [12], extended the 2-machines Johnson's algorithm by converting a n-jobs, m-machines (m>2) into p number of 2-machines n-jobs surrogate problems where p=m-1. Each of the surrogate problems was solved using Johnson rule. Palmer [13] presented a heuristic for solving n-jobs m-machines PFSP. The method was extended in Hundal and Rajgopal [14]. Nawaz et al. [15], developed the Nawaz-Enscore-Ham (NEH) heuristic in [15] for PFSP.

The problem of heuristic-based methods being trapped in local optimum solutions, has however caused researchers' drift towards metaheuristics-based methods. The tabu search and simulated annealing (SA) were two early metaheuristics applied on PFSPs [16−19]. The two methods are single-solution search methods that iteratively improve on a solution at a time.

Many population-based metaheuristics have also been used to solve PFSP. The genetic algorithm (GA) and differential evolution are among them [20−22]. The swarm intelligence methods such as the ant colony [23, 24] and the particle swarm optimization (PSO) have also been used [25, 26]. A review of application of metaheuristics to FSP can be found in [3, 27-29]. Although a large number of optimization methods have been proposed for solving DOPs, none of them is able to solve all instances of this problem. Each of them has a DOP in which it is ineffective or inapplicable. This was the motivation for the study reported in this paper.

2.Materials and methods

In this section the continuous optimization method [1] and its adaptation to discrete domain have been presented. The continuous optimization method is population-based and employs a metaheuristic to

guide its search. The symbols used in the formulation of the algorithms have been shown in *Table 1*.

Table 1 The symbols used in the algorithms

Symbol	Description
D	Dimension of the problem
P	Population size
N	Number of iterations (over all the particles)
$distance(\underline{u},\underline{v})$	Geometric distance of $\underline{v} = (v_1, v_2, \ldots, v_D)$ from $\underline{u} = (u_1, u_2, \ldots, u_D)$
$\underline{minx}$	The ($minx_1$, $minx_2$, ….., $minx_D$)
$\underline{Maxx}$	The vector ($maxx_1$, $maxx_2$, ….., $maxx_D$)
$\underline{x_i^k}$:	The ith particle $(x_{i,,1}^k, x_{i,,2}^k, \ldots, x_{i,,D}^k)$ position on the kth round of iterations
$\underline{LB_i^k}$	The ith particle local best position, $(LB_{i,,1}^k, LB_{i,,2}^k, \ldots, LB_{i,,D}^k)$ on the kth round of iterations
$\underline{IB^k}$	The best particle result among all the particles only on the kth iteration
$\underline{GB^k}$:	The global best position, $(GB_1^k, GB_2^k, \ldots, GB_D^k)$ of all the particles up to the kth round of iterations
fitValue($\underline{z}$)	The fitness value of $\underline{z}$
ϵ	The minimum distance allowed between a particle and the current global position
cRate	A real value in the interval [0,1] to be supplied by user
c_0, c_1, c_2	Non-negative, user-supplied constants
rand()	A random number generated in [0,1]

Algorithm 1

Initialization step

(a) Initialize randomly the positions $\underline{x_i}^{(0)}$ of all the particles in the population:

$\underline{x_i}^{(0)} = \underline{minx} + rand() \times (\underline{maxx} - \underline{minx})$ for i=1,2…,P

(b) Set the global best position $\underline{GBest}^0$ to the particle position with the best fitness value.

Iterative step

for k=1, 2,…….N do the following

for (i=1 ,….., P) do the following

{(α)Update $\underline{x_i}^k$ to obtain $\underline{x_i}^{k+1}$:

(a) (i) for (j=0, 1 … to D) do

if (rand()>cRate)then $u_j = GB_j^k$ (1)

else $u_j = c_1\ LB_{i,j}^k + c_2\ GB_j^k$ (2)

(ii) If (u_j is not in the interval [$minx_j$, $maxx_j$]) then u_j= $minx_j$ + rand()×($maxx_j$ - $minx_j$)

(b) for (j=0, 1 … to D) do

{(i) $v_j = x_{ij}^k + rand() \times c_0 \times (GB_j^k - x_{ij}^k)$ (3)

(ii) If ($v_j$is not in the interval [$minx_j$, $maxx_j$]) then v_j= $minx_j$ + rand()×($maxx_j$ - $minx_j$)}

(c) if (fitValue($\underline{v}$) >fitValue($\underline{u}$)) then set

$\underline{x_i}^{k+1} = \underline{v}$

else set $\underline{x_i}^{k+1} = \underline{u}$

(d) if(distance($\underline{x_i}^{k+1}$, $\underline{GB}^k$) < ϵ) then

$\underline{x_i}^{(k+1)} = \underline{minx} + rand() \times (\underline{maxx} - \underline{minx})$

(β) Update global best position fitness value:

if (fitValue($\underline{GB}^k$) < fitValue($\underline{x_i}^{k+1}$)) then $\underline{GB}^{k+1} = \underline{x_i}^{k+1}$

else $\underline{GB}^{k+1} = \underline{GB}^k$}

Output the current global best position,$\underline{GB}^N$, and its fitness value, fitValue($\underline{GB}^N$).

This algorithm was tested in [1] on many benchmark functions and found to perform better than the algorithms considered in the paper.

2.1The continuous optimization problem

Using the symbols on *Table 1*, the hybrid continuous optimization method in [1] can be stated as follows:

This encouraged us to investigate its performance on DOPs and on PFSP in particular.

2.2Discrete version of the algorithm

To adapt the above algorithm for discrete optimization, the Clerc approach [30, 31] is used to discretize Equation 2,

$\underline{u}_i^{k+1} = c_1 \times \underline{LB}_i^k + c_2 \times \underline{GB}^k$,

Which defines a potential particle position $\underline{u}_i^{k+1}$, as a linear combination of $\underline{LB}_i^k$ and $\underline{GB}^k$.

To accomplish this, the equation, is transformed into the form

$\underline{u}_i^{k+1} = (c_1 + c_2)\,\underline{x}_i^k + c_1 \times (\underline{LB}_i^k - \underline{x}_i^k) + c_2 \times (\underline{GB}^k - \underline{x}_i^k)$

that is subsequently generalized to

$\underline{u}_i^{k+1} = c_0'\underline{x}_i^k + c_1' \times (\underline{LB}_i^k - \underline{x}_i^k) + c_2' \times (\underline{GB}^k - \underline{x}_i^k)$

Where c_0' c_1', c_2', are values in the interval [0, 1].

Through computational experiment, we observe a better result when the local best, $\underline{LB}_i^k$, is replaced with the instant best position, $\underline{IB}^k$ (the best particle position within the k[th] iteration). Consequently we replace $\underline{LB}_i^k$, with $\underline{IB}^k$ in the last equation to have

$\underline{u}_i^{k+1} = c_0'\underline{x}_i^k + c_1' \times S(\underline{IB}^k, \underline{x}_i^k) + c_2' \times S(\underline{GB}^k, \underline{x}_i^k)$ (4)

Where the symbol $S(\underline{p}, \underline{q})$ (for any two positions $\underline{p}, \underline{q}$) denotes $\underline{p} - \underline{q}$.

Following Clerc's approach, the operators -, +, × in Equation 4 are however interpreted as follows for discrete domain.

$S(\underline{p}, \underline{q})$ = $\underline{p} - \underline{q}$: is a collection of swaps of entries of $\underline{q}$ required to make $\underline{q}$ equal $\underline{p}$.

c× $S(\underline{p}, \underline{q})$: a subset of $S(\underline{p}, \underline{q})$ of size c| $S(\underline{p}, \underline{q})$| picked randomly (where | $S(\underline{p}, \underline{q})$| denotes the size of set $S(\underline{p}, \underline{q})$). (5b)

$S(\underline{p}, \underline{q}) + S(\underline{u}, \underline{v})$: produces the union of $S(\underline{p}, \underline{q})$ and $S(\underline{u}, \underline{v})$ devoid of duplicates (5c)

p+S(×,×):returns the position produced on performing the swaps in S(×,×) on position p. (5d)

$c'^{\underline{p}} =$

$\begin{cases} mutate\ \underline{p}\ by\ swap\ any\ two\ entries\ of\ it\ if\ rand(\ ) > c' \\ \underline{p} \quad otherwise \end{cases}$ (5e)

To increase the exploration of the local area around $\underline{GB}^k$, $\underline{x}_i^k$, Equation 3 in the algorithm above is expanded as

$\underline{v}_i^{k+1} = (1 - c)\underline{x}_i^k + c \times \underline{GB}^k$

(Where= $rand() * c_0$) and implemented as follows to produce a candidate position $\underline{v}_i^{k+1}$.

$v_{ij}^{k+1} = \begin{cases} GB_j^k & if\ rand() > c \\ x_{ij}^k & otherwise \end{cases}$, $\underline{v}_i^{k+1} = v_{i1}^{k+1}, v_{i2}^{k+1}, \ldots, v_{iD}^{k+1})$ (6)

With $\underline{v}_i^{k+1}$ repaired where infeasible, i.e $\underline{v}_i^{k+1} = repair(\underline{v}_i^{k+1})$).

The two positions, $\underline{u}_i^{k+1}$ and $\underline{v}_i^{k+1}$, (in Equations 4 and 6) are then compared with respect to their fitness values and the better is chosen for $\underline{x}_i^{k+1}$.

The guiding principle adopted in the formulation above is to make each particle i, retain part (i.e some entries) of the current global best optimum, $\underline{GB}^k$ in its next position, $\underline{x}_i^{k+1}$. The part of $\underline{GB}^k$ retained is controlled by the user-supplied parameter c. Observe that $\underline{x}_i^{k+1} = \underline{GB}^k$ when c=1 while $\underline{x}_i^{k+1} = \underline{x}_i^k$ when c=0.

By way of illustration, consider the case when the number of jobs is 7 (with the jobs numbered 0,1..6) and suppose $\underline{IB}^k$, $\underline{x}_i^k$, $\underline{GB}^k$ contain the entries shown in the arrays below (the entries being the job numbers).

	0	1	2	3	4	5	6
$\underline{IB}^k$:	2	1	6	0	5	4	3

	0	1	2	3	4	5	6
$\underline{GB}^k$:	3	0	2	6	4	5	1

	0	1	2	3	4	5	6
$\underline{x}_i^k$:	6	1	2	5	4	0	3

then

$S(\underline{IB}^k, \underline{x}_i^k) = \underline{IB}^k - \underline{x}_i^k = \{(0,2), (3,4), (3,5)\}$,

$S(\underline{GB}^k, \underline{x}_i^k) = \underline{GB}^k - \underline{x}_i^k = \{(0,6), (1,5), (3,6), (5,6)\}$

(Where (i, j) specifies that the entries of the i[th] and j[th] components of the arrays in S (. ,.) be swapped)

If $c_1' = c_2' = 0.5$ then $c_1' \times |S(\underline{IB}^k, \underline{x}_i^k)| = 0.5 * 3 = 1.5 => 2$, $c_2' * |S(\underline{GB}^k, \underline{x}_i^k)| = 0.5 * 4 = 2$ indicating that two elements of $S(\underline{IB}^k, \underline{x}_i^k)$ and of $S(\underline{GB}^k, \underline{x}_i^k)$ be chosen at random for $c_1' \times S(\underline{IB}^k, \underline{x}_i^k)$ and $c_2' \times S(\underline{GB}^k, \underline{x}_i^k)$ respectively. For example with the choice,

$c_1' * S(\underline{IB}^k, \underline{x}_i^k) = \{(0,2), (3,4)\}$

$c_2' * S(\underline{GB}^k, \underline{x}_i^k) = \{(1,5), (5,6)\}$

We would have (using Equation 5c),
$S(\times,\times) = c_1' \times S(\underline{IB}^k\ ,\ \underline{x_i}^k\ ) +\ c_2' \times S(\ \underline{GB}^k,$
$\underline{x_i}^k\ ) = \{(0,2), (3,4)\ ,\ (1,5), (5,6)\}$

If $c_0' = 0.5$ for example and suppose the random generator, rand () in Equation 5e returns 0.9 then

$c_0'\underline{x_i}^k$:

6	0	2	5	4	1	3

[The contents of the second and sixth components of $\underline{x_i}^k$ are chosen at random for exchange to produce the result.]

Equation 4 (with definition in Equation 5d) produces.
$\underline{u_i}^{k+1} = c_0'\underline{x_i}^k + c_1' \times S(\underline{IB}^k, \underline{x_i}^k) + c_2' \times S(\underline{GB}^k, \underline{x_i}^k)$
$= c_0'\underline{x_i}^k\ + S(*,*)$:

2	1	6	4	5	3	0

If c=0.5 and the random generator, rand(), in Equation (5e) generates the numbers 0.5, 0.1, 0.2, 0.7 , 0.6, 0.1, 0.9, 0.4 in that order then the entries of $\underline{v_i}^{k+1}$ computed using Equation (5e) yields:

$\underline{v_i}^{k+1}$:

6	1	2	6	4	5	3

$\Longrightarrow$ *repaired to have*

$\underline{v_i}^{k+1}$:

6	1	2	0	4	5	3

The better of $\underline{u_i}^{k+1}$ and $\underline{v_i}^{k+1}$ in terms of fitness value is then chosen for $\underline{x_i}^{k+1}$. In summary, the discrete version of the algorithm above can be stated as follows.

Algorithm 2

Input: input values for c, c_0', c_1', c_2', D (the dimension denoting the number of jobs), P (population size), N (number of iterations to perform) and the problem benchmark instance.

Initialization step

(a)Initialize the particles' positions $\underline{x_i}^{(0)}$ (i=1,2..P) with feasible solutions

(b)Compute the fitness values of the particles' positions, $\underline{x_i}^{(0)}$ (i=1..P) and set the global and instant best positions $\underline{GBest}^0$, $\underline{IB}^0$ equal the position of the particle that has the best fitness value.

Iterative step

On the k^{th} iteration (k=0, 1,…….N-1) do the following

{for the i^{th} particle position, $\underline{x}_i^k$ (i=1 ,….., P) do the following

{1: Obtain $\underline{x}_i^{k+1}$ from $\underline{x}_i^k$ $\underline{IB}^k$ and $\underline{GB}^k$ as follows

(a) Compute $\underline{u_i}^{k+1}$ from equation (4) using the definition of operators -, + and × given in equations 5a…5e. as illustrated in the example above

(b) Compute $\underline{v_i}^{k+1}$ using equation (6)

(c) Repair $\underline{v_i}^{k+1}$ (if not a feasible solution) using a repair function

(d) Compute the fitness values of positions $\underline{u_i}^{k+1}$ and $\underline{v_i}^{k+1}$ and set $\underline{x_i}^{k+1}$ equal to the position that has the better fitness value among the two.

(e) if $\underline{x_i}^{k+1}$ equals $\underline{GB}^k$ then generate a feasible solution for $\underline{x_i}^{(k+1)}$ randomly.

2: Update the instant global best position $\underline{IB}^{k+1}$ (by setting $\underline{IB}^{k+1}$ equals $\underline{x_i}^{k+1}$ if the fitness value of $\underline{x_i}^{k+1}$ is better)}

Update the global best position, $\underline{GB}^{k+1}$, (by setting $\underline{GB}^{k+1}$ equals $\underline{IB}^{k+1}$ if the fitness value of $\underline{IB}^{k+1}$ is better than that of the current $\underline{GB}^{k+1}$) }

Output the current global best position, $\underline{GB}^N$, and its fitness value.

While the initialization of the particles' positions (required in the initialization step of the algorithm) might be done differently, the procedure below describes the way it was done in our computational experiment.

Procedure for generating initial positions

(a)Fill each individual,$\underline{x_i}^{(0)}$, with integers 1.2.. D (i.e set position $\underline{x_i}^{(0)}$=(1,2,…,, D) for i=1,…P)

(b)Alter the entries of each individual , $\underline{x_i}^{(0)}$ (i=1..P) to make $\underline{x_i}^{(0)}$ (i=1,..D) distinct and scattered (and not just neighbours): This is done the following way in the algorithm.

For each individual $\underline{x_i}^{(0)}$ (i=1,..n) do the following for each of its component, j' (j=1,..D)

{Generate a random integer, k, in the range [j+1, Interchange the entries of the j^{th} and k^{th} components of $\underline{x_i}^{(0)}$ }

The motive behind step (b) is to ensure a wide spread of the initial particles' positions to enable "good" solution be reached and possibly at the shortest possible time. The random feasible solution mentioned in step 1 (e) of the algorithm is generated using this procedure for generating initial positions but only for one position. The repair of an infeasible solution mentioned in step 1c is done by replacing each duplicate copy of an entry with a distinct job number.

3.Results

In an attempt to investigate its performance on DOPs, the algorithm was coded in Java using NetBeans 5.0 and tested on the flowshop problem using Taillard benchmark instances [2]. In all, the population size (P) was set at 20 and the number of iterations (N) varied from 2000 to 30,000. The parameters c, c_0', c_1', c_2', were set to 0.5, 1.0, 0.8 and 0.8 respectively. The algorithm was run 20 times on each instance and the best result (among them) presented on *Tables 2, 4 and 5* (for each instance). The Taillard instances selected for the test were those used for the existing algorithms specified on the tables. Column 1 of each table contains the instance tags while the last column contains the result from our algorithm.

Column 2 of *Table 2* contains the bounds on the optimal solutions for the instances while column 3 contains the result. It is obtained by Ramanan et al. [32] using their algorithm, PSO-NEH-VNS (a hybrid of PSO, NEH, and neighbourhood search (NS)).

Ramanan et al. [32] compared their result with those from Campbell's CDS heuristic, NEH heuristic, PSO, PSO-NEH, and PSO-VNS. They found their results better on these benchmark instances.

Column 4 of *Table 2* shows the result obtained by Gupta and Chauhan [33] (with their heuristic). Their result is better than those from Palmer's heuristic and Dannenbring's rapid access heuristics on the flowshop problem [33].

The last column shows the relative difference percentage (Rel_Diff%). It shows the relative difference percentage of our approach (in column 5) from a known solution for each instance.

Rel_Diff% was computed using the formula:

$$\text{Rel\_Diff\%} = \frac{best\_known\_solution - algorthim\_solution}{best\_known\_solution} \times 100 \quad (9)$$

Where the best_known_solution is taken to be the upper-bound (in column 2) on the solution of each instance.

Table 2 Comparison of makespans obtained from Ramanan et al. [32] and Gupta and Chauhan [33] methods with those from the algorithm above on some flowshop Taillard benchmark problems

Instance number {Size}	Solution interval bound	Ramanan et al. [32]	Gupta and Chauhan [33]	Algorithm proposed	
				Makespan	Rel_Diff%
ta001 {20x5}	[1232,1278]	1278	1367	1278	0
ta002 {20x5}	[1290,1359]	1365	1432	1359	0
ta003 {20x5}	[1073,1081]	1100	1162	1081	0
ta004 {20x5}	[1268,1293]	1309	1402	1293	0
ta005 {20x5}	[1198,1236]	1250	1300	1235	-0.081
ta011 {20x10}	[1448,1582]	1586	1658	1583	0.063
ta012 {20x10}	[1479,1659]	1684	1802	1660	0.060
ta013 {20x10}	[1407,1496]	1521	1621	1502	0.401
ta014 {20x10}	[1308,1378]	1399	1548	1378	0
ta015 {20x10}	[1325,1419]	1450	1638	1419	0
ta021 {20x20}	[1911,2297]	2330	2559	2297	0
ta022 {20x20}	[1711,2100]	2111	2303	2106	0.286
ta023 {20x20}	[1844,2326]	2342	2567	2336	0.43
ta024 {20x20}	[1810,2223]	2248	2458	2235	0.54
ta025 {20x20}	[1899,2291]	2302	2454	2299	0.349
ta031 {50x5}	[2712,2724]	2723	2800	2724	0
ta032 {50x5}	[2808,2834]	2843	3015	2838	0.141
ta033 {50x5}	[2596,2621]	2631	2702	2621	0
ta034 {50x5}	[2740,2751]	2762	2845	2753	0.073
ta035 {50x5}	[2837,2863]	2864	2960	2864	0.035
ta041 {50x10}	[2907,3025]	3059	3468	3046	0.694
ta042 {50x10}	[2821,2892]	2934	3174	2918	0.9
ta043 {50x10}	[2801,2864]	2932	3180	2876	0.419
ta044 {50x10}	[2968,3064]	3115	3353	3078	0.457

Instance number {Size}	Solution interval bound	Ramanan et al. [32]	Gupta and Chauhan [33]	Algorithm proposed	
				Makespan	Rel_Diff%
ta045 {50x10}	[2908,2986]	3052	3356	3021	1.172
ta051 {50x20}	[3480,3875]	4010	4256	3926	1.316
ta052 {50x20}	[3424,3715]	3864	4255	3787	1.938
ta053 {50x20}	[3351,3668]	3808	4104	3730	1.690
ta054 {50x20}	[3336,3752]	3844	4203	3790	1.013
ta055 {50x20}	[3313,3635]	3815	4091	3691	1.541
ta061 {100x5}	[5437,5493]	-	5673	5493	0
ta062 {100x5}	[5208,5268]	-	5380	5290	0.418
ta063 {100x5}	[5130,5175]	-	5452	5211	0.696
ta064 {100x5}	[4963,5014]	-	5148	5021	0.14
ta065 {100x5}	[5195,5250]	-	5286	5253	0.057
ta071 {100x10}	[5759,5770]	-	6153	5810	0.693
ta081 {100x20}	[5851,6286]	-	6957	6403	1.861
ta091 {200x10}	[10816,10868]	-	11258	10940	0.662
ta101 {200x20}	[10979,11294]	-	12587	11463	1.5

Table 3 presents the schedule for 20 jobs, 5 machines (i.e. ta005). The number in the curly bracket is the processing time required by the job and the number before the bracket is the time of the job starts on the machine. Consequently the sum of the two numbers gives the time the job leaves the machine. The first column contains the sequence number of the job. For example, job 11 is the first job executed on machine 1 while job 4 is the second job executed on the machine. Recall that the ordering of the jobs for execution is the same for all the machines.

Kwiecein and Filipowicz [34] also compared the results of the firefly algorithm (FA) and cockroach algorithm (CA) on some Taillard benchmark instances.

Figure 1 shows the comparison of makespan values presented in Kwiecein and Filipowicz [34] for FA and CA with those from the algorithm proposed. Alg shows the result of our approach. The graph is a plot of the makespan values against problem instances for the algorithms.

The result of our algorithm is also compared with those other algorithms considered by Ying and Liao [24]. It is shown in *Figure 2*. The labels are ant colony optimization (ACO), Palmer, GA, SA and (NS). *Figure 2* presents the Comparison of Rel_Diff% obtained from the algorithms reported in [24] and the Alg.

Table 3 Job sequencing (scheduling) for ta005 problem on *Table 1*

Seq. No	Job No.	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5
1	11	0{3}	3{32}	35{38}	73{14}	87{87}
2	4	3{14}	35{34}	73{16}	89{19}	174{22}
3	3	17{42}	69{65}	134{30}	164{70}	234{84}
4	18	59{42}	134{59}	193{9}	234{91}	325{33}
5	2	101{16}	193{8}	202{32}	325{6}	358{56}
6	8	117{46}	201{2}	234{95}	331{57}	414{62}
7	15	163{59}	222{95}	329{39}	388{89}	477{64}
8	9	222{41}	317{7}	368{21}	477{60}	541{61}
9	16	263{34}	324{48}	389{97}	537{37}	602{62}
10	1	297{86}	383{92}	486{93}	579{47}	664{48}
11	12	383{72}	475{14}	579{4}	626{90}	716{99}
12	5	455{92}	547{6}	583{95}	716{97}	815{51}
13	10	547{78}	625{85}	710{74}	813{62}	875{10}
14	14	625{53}	710{59}	784{62}	875{12}	887{91}
15	13	678{95}	773{74}	847{31}	887{76}	978{40}
16	6	773{67}	847{42}	889{58}	963{41}	1018{43}
17	17	840{66}	906{37}	947{57}	1004{35}	1061{53}
18	0	906{61}	967{27}	1004{42}	1046{13}	1114{55}
19	19	967{63}	1030{4}	1046{54}	1100{69}	1169{16}
20	7	1030{77}	1107{39}	1146{12}	1169{1}	1185{50}

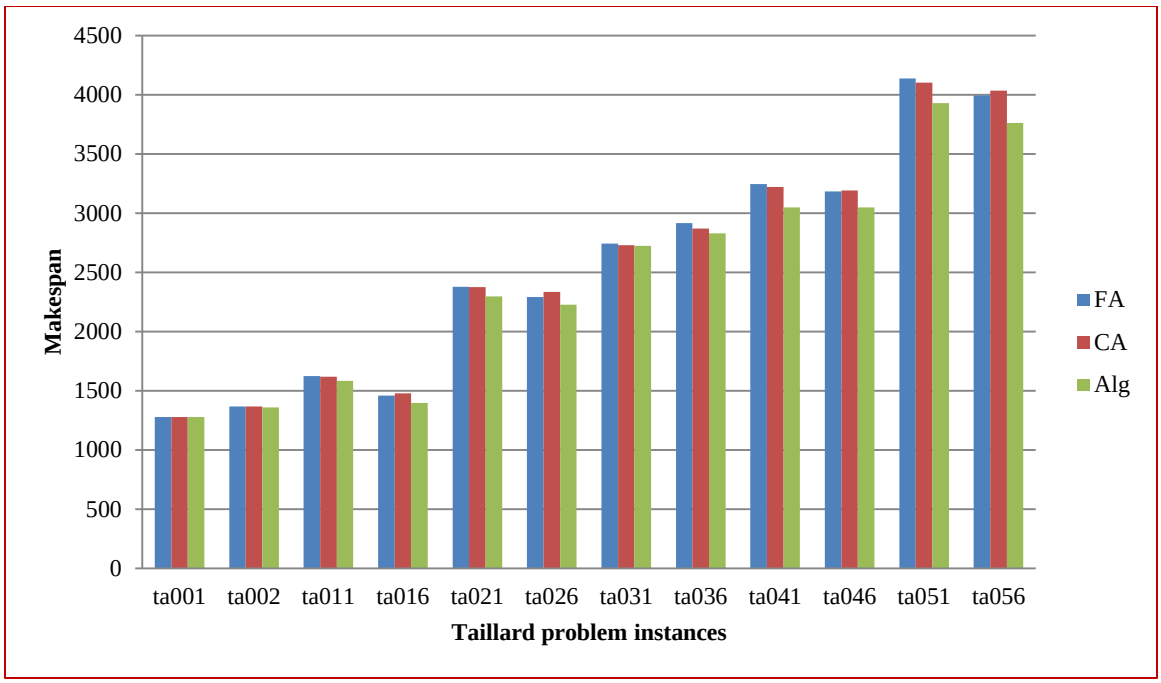

Figure 1 Makespan values obtained from FA, CA and the proposed approach (Alg) against the Taillard benchmark instances ta001−ta056 indicated on the horizontal axis

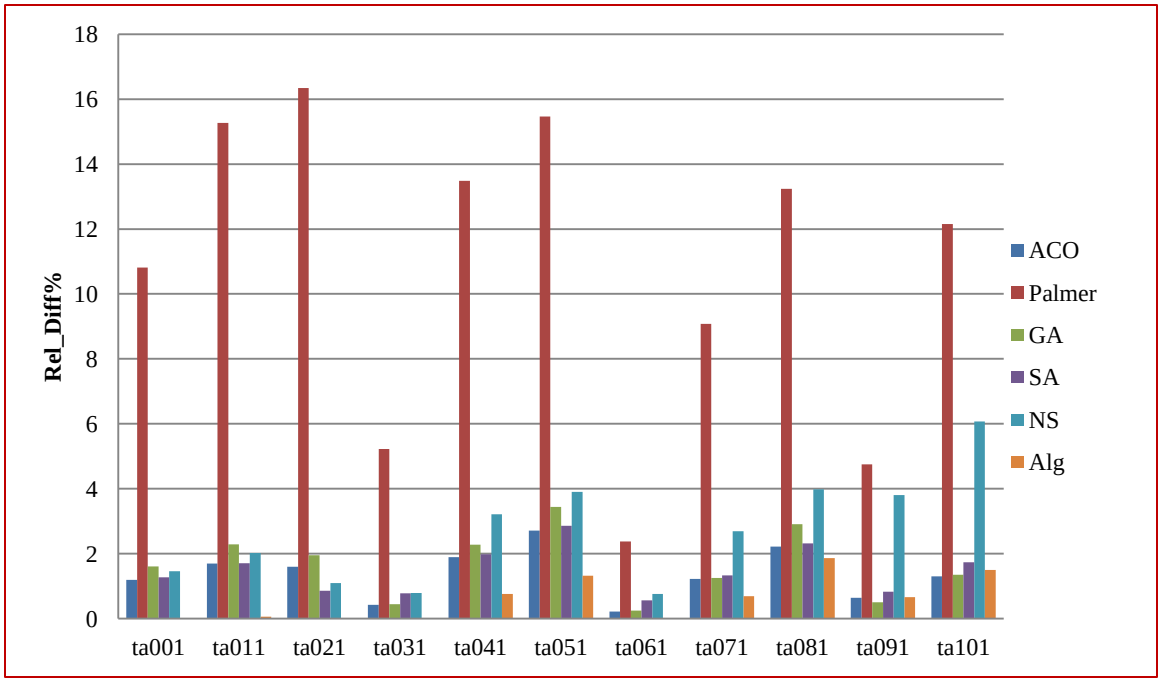

Figure 2 Comparison of Rel_Diff% values of ACO, Palmer, GA, SA, NS algorithms and the proposed approach (Alg) against the Taillard benchmark instances ta001, ..,ta101

4.Discussion

The algorithm above is proposed for DOPs and tested on the PFSP with makespan as the performance measure. The Taillard instances with 200 jobs and 20 machines were used in the test. Except for ta005 and the instances with 5 machines, the results for all other instances, fall out of the interval bound shown in *Table 2*. The solutions produced for instances with 5

machines coincide with the upper limits of the bounds or very close to them. It is observed that the number of jobs and machines adversely affect the quality of solution the algorithm produces. The solution moves further away from the bound as the number of machines and jobs increases. The algorithm in general produced results comparable with other algorithms considered in the experiment.

The computational complexity of an algorithm for solving a problem is the maximum number of computational steps needed to solve the problem (in this case, the computational steps needed to obtain an optimal solution). The metaheuristic based algorithm produces only "good" solution (not necessarily optimal and not precisely defined) and in some cases may run indefinitely without attaining an optimal solution or a preset bound on the solution. This makes it difficult to come up with computational complexity of such algorithm. In fact, in many cases the number of iterations is usually supplied to terminate such algorithm to avoid running into an infinite loop. *Table 4* shows the major operations in the algorithm and their cost.

In this table D denotes the number of jobs, P shows the population size and M indicates the number of machines.

Since the dominant operations are the repair and fitness function evaluation, the computational cost of the algorithm for n number of iterations may be taken as
$O(n\times (2 \times D^2 \times P + 2 \times D \times M \times P)) = O(n \times (D^2 \times P + D \times M \times P))$.

The experimental running time of an algorithm depends on the processor speed, and code (i.e. the structure of the program. For instance; A program with many function calls and classes solving a problem may take more time (due to the over-head cost of loading classes and function calling) than the one with less numbers of function call and classes solving the same problem. The randomness of the algorithm (being of a metaheuristic type) makes the running time vary from one running of the program to another. For these reasons *Table 5* presents the running time obtained for the number of iterations specified with the results (i.e. Rel_Diff) obtained in some instances.

Table 4 Running cost per one round of iteration

Operations	**Cost of operation**	**Maximum number undertaken in one iteration**	**Cost in one iteration**
Repair	$O(D^2)$	1(for each particle position, total 1xP)	$O(D^2\times P)$
Test of equality undertaken in step 1(e)	O(D)	1(for each particle position, total 1xP)	O(D×P)
Evaluation of fitness function, 1(d)	O(D×M)	2(for each particle position, total 2xP)	O(D×M×2P)
Computation in step 1(a, b)	$O(D^2)$	1(for each particle position, total 1xP)	$O(D^2\times P)$
Updating operations (i.e. for $\underline{IB}^{k+1}$, $\underline{GB}^{k+1}$ in step 2 and the last step of the algorithm	O(D)	2(for each particle position, total 2xP)	O(2×D×P)
Position generated randomly in step 1(e)	O(D)	1(for each particle position, total 1xP)	O(D×P)
Initialization step (if the initialization procedure is adopted)	O(P×D)	-	-
Loading instances	O(D×M)	-	-
Display	O(D×M)	-	-

Table 5 Running time of the algorithm for some instances for specified number of iterations

Problem instances	**Number of iterations**	**Rel_Diff**	**Time**
ta001 (20x5)	2000	0	7 sec
ta011 (20x10)	2000	0.063	11 sec
ta021 (20x20)	8000	0	1 min 13 sec
ta031(50x5)	2000	0	19 sec
ta041(50x10)	10000	0.694	4 min 31sec
ta061 (100x5)	10000	0	3 min 9 sec
ta071(100x10)	10000	0.693	4 min 40 sec
ta091(200x10)	10000	0.662	9 min 34 sec
ta101 (200x20)	20000	1.5	45 min 58 sec

5.Conclusion and future direction

In this work the optimization technique presented in [1] for continuous domain is adapted to handle discrete optimization problems using Clerc approach devised for the PSO algorithm. The addition, subtraction and multiplication operators in the algorithm meanings are different from their normal meaning and the resulting algorithm applied to the traveling salesman problem. The adapted algorithm presented above is applied to the permutation flowshop problem using the Taillard benchmark instances. The result obtained is compared with other algorithms and the presented approach found to be better.

The following future works are suggested.

- It can be tested on the benchmark instances other than used in these algorithms discussed.
- It can be tested on benchmark instances other than those from Taillard.
- In future other performance measures can be used in place of makespan.
- Investigation of the algorithm's performance on some other DOPs especially real-world problems.

Acknowledgment

None.

Conflicts of interest

The authors have no conflicts of interest to declare.

References

[1] Shola PB, Asaju LB. An algorithm for continuous optimization problems using hybrid particle updating method. Indonesian Journal of Electrical Engineering and Computer Science. 2016; 3(1):164-73.
[2] Taillard E. Benchmarks for basic scheduling problems. European Journal of Operational Research. 1993; 64(2):278-85.
[3] Bagchi TP, Gupta JN, Sriskandarajah C. A review of TSP based approaches for flowshop scheduling. European Journal of Operational Research. 2006; 169(3):816-54.
[4] Garey MR, Johnson DS, Sethi R. The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research. 1976; 1(2):117-29.
[5] Ignall E, Schrage L. Application of the branch and bound technique to some flow-shop scheduling problems. Operations Research. 1965; 13(3):400-12.
[6] Lomnicki ZA. A "branch-and-bound" algorithm for the exact solution of the three-machine scheduling problem. Operations Research. 1965; 16(1):89-100.
[7] McMahon GB, Burton PG. Flow-shop scheduling with the branch-and-bound method. Operations Research. 1967; 15(3):473-81.
[8] Horowitz E, Sahni S. Fundamentals of computer algorithms. Computer Science Press; 1978, p.626.
[9] Johnson SM. Optimal two-and three-stage production schedules with setup times included. Naval Research Logistics. 1954; 1(1):61-8.
[10] Sen T, Dileepan P, Gupia JN. The two-machine flowshop scheduling problem with total tardiness. Computers & Operations Research. 1989; 16(4):333-40.
[11] Wang B, Li T, Shi C, Wang H. Scheduling two-machine flowshop with limited waiting times to minimize makespan. Indonesian Journal of Electrical Engineering and Computer Science. 2014; 12(4):3131-9.
[12] Campbell HG, Dudek RA, Smith ML. A heuristic algorithm for the n job, m machine sequencing problem. Management Science. 1970; 16(10): 630-7.
[13] Palmer DS. Sequencing jobs through a multi-stage process in the minimum total time-a quick method of obtaining a near optimum. Journal of the Operational Research Society. 1965; 16(1):101-7.
[14] Hundal TS, Rajgopal J. An extension of Palmer's heuristic for the flow shop scheduling problem. International Journal of Production Research. 1988; 26(6):1119-24.
[15] Nawaz M, Enscore EE, Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega. 1983; 11(1):91-5.
[16] Grabowski J, Wodecki M. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. Computers & Operations Research. 2004; 31(11):1891-909.
[17] Ekşioğlu B, Ekşioğlu SD, Jain P. A tabu search algorithm for the flowshop scheduling problem with changing neighborhoods. Computers & Industrial Engineering. 2008; 54(1):1-11.
[18] Ishibuchi H, Misaki S, Tanaka H. Modified simulated annealing algorithms for the flow shop sequencing problem. European Journal of Operational Research. 1995; 81(2):388-98.
[19] Parthasarathy S, Rajendran C. A simulated annealing heuristic for scheduling to minimize mean weighted tardiness in a flowshop with sequence-dependent setup times of jobs-a case study. Production Planning & Control. 1997; 8(5):475-83.
[20] Chen CL, Vempati VS, Aljaber N. An application of genetic algorithms for flow shop problems. European Journal of Operational Research. 1995; 80(2):389-96.
[21] Chaudhry IA, Khan AM. Minimizing makespan for a no-wait flowshop using genetic algorithm. Sadhana. 2012; 37(6):695-707.
[22] Čičková Z, Števo S. Flow shop scheduling using differential evolution. Management Information Systems. 2010; 5(2):8-13.
[23] Yagmahan B, Yenisey MM. A multi-objective ant colony system algorithm for flow shop scheduling problem. Expert Systems with Applications. 2010; 37(2):1361-8.

[24] Ying KC, Liao CJ. An ant colony system for permutation flow-shop sequencing. Computers & Operations Research. 2004; 31(5):791-801.
[25] Tasgetiren MF, Liang YC, Sevkli M, Gencyilmaz G. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. European Journal of Operational Research. 2007; 177(3):1930-47.
[26] Ponnambalam SG, Jawahar N, Chandrasekaran S. Discrete particle swarm optimization algorithm for flowshop scheduling. In Particle Swarm Optimization. InTech; 2009, p.397-422.
[27] Reza Hejazi S, Saghafian S. Flowshop-scheduling problems with makespan criterion: a review. International Journal of Production Research. 2005; 43(14):2895-929.
[28] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research. 2005; 165(2):479-94.
[29] Tyagi N, Varshney RG, Chandramouli AB. Six decades of flowshop scheduling research. International Journal of Scientific & Engineering Research. 2013; 4(9):854-64.
[30] Clerc M. Discrete particle optimization illustrated by the traveling salesman problem.2000.
[31] Clerc M. Discrete particle swarm optimization, illustrated by the traveling salesman problem. In new optimization techniques in engineering 2004 (pp. 219-39). Springer Berlin Heidelberg.
[32] Ramanan TR, Iqbal M, Umarali K. A particle swarm optimization approach for permutation flow shop scheduling problem. International Journal for Simulation and Multidisciplinary Design Optimization. 2014; 5:1-6.
[33] Gupta A, Chauhan S. A heuristic algorithm for scheduling in a flow shop environment to minimize makespan. International Journal of Industrial Engineering Computations. 2015; 6(2):173-84.
[34] Kwiecień J, Filipowicz B. Comparison of firefly and cockroach algorithms in selected discrete and combinatorial problems. Bulletin of the Polish Academy of Sciences Technical Sciences. 2014; 62(4):797-804.

Dr. Peter B. Shola received his B.SC. in Mathematics from Ahmadu Bello University Nigeria in 1979 and M.Sc and Ph.D from University of Essex , UK in 1984 and 1991 respectively. He is a lecturer in the Department of Computer Science, University of Ilorin Nigeria. His research interests include Evolutionary Algorithms, Data Mining and Computational Fluid Dynamics.
Email: shola.bp@unilorin.edu.ng

Dr. Asaju La'aro Bolaji received his B.Tech. in Physics from Federal University of Technology, Minna, Nigeria and the M.Sc. in Mathematics/ Computer Science Option from the University of Ilorin, Nigeria in 2001 and 2006 respectively. He was awarded PhD degree in Computer Science at Universiti Sains Malaysia in April 2014. He is a member of the Academic staff in the Department of Computer Science, Federal University Wukari, Wukari, Taraba State, Nigeria. His research interests include Evolutionary Algorithms, Nature-Inspired Computation, and Complex Optimization Problems.
Email: lbasaju@fuusukavi.edu.ng