An Estimation of Distribution Algorithm for Flowshop Scheduling with Limited Buffers

Mansour Eddaly, Bassem Jarboui, Patrick Siarry, and Abdelwaheb Rebaï

Abstract. Most of the works that address the flowshop scheduling problem presume unlimited buffers between successive machines. However, with the advent of new technologies in the manufacturing systems, limited capacity storage between machines has become profitable. Aimed at makespan minimization, the flowshop scheduling problem with buffer constraints is NP-hard in the strong sense. Therefore, several approximate algorithms have been proposed in the literature. In this chapter, we propose an Estimation of Distribution Algorithm for solving a flowshop scheduling problem with buffer constraints. The main characteristics of the problem, such as the order of jobs and similar blocks of jobs in the sequence, are taken into account while building the probabilistic model. In order to enrich the search procedure of the algorithm, a skewed variable neighbourhood search algorithm is embedded into it, restricted by a calculated probability which depends on the quality of the created offspring. The computational results show that our algorithm outperforms genetic algorithm and particle swarm algorithm, and can obtain several optimal solutions in a short time.

1 Introduction

In a classical flowshop scheduling problem (FSP), each job j $(j = 1, 2, ..., n)$ must be processed on every machine i $(i = 1, 2, ..., m)$ and all the jobs have to pass through all the machines following the same route. In such problems, an infinite buffer size between every two successive machines is assumed. Due to the modernization of

Mansour Eddaly, Bassem Jarboui, and Abdelwaheb Rebaï
FSEGS, route de l'aéroport km 4.5, B.P. No. 1088, Sfax 3018, Tunisie
e-mail: `eddaly.mansour@gmail.com`, `bassem_jarboui@yahoo.fr`, `abdelwaheb.rebai@fsegs.rnu.tn`

Patrick Siarry
LiSSi, Université de Paris 12, 61 avenue du Général de Gaulle, 94010 Créteil, France
e-mail: `siarry@univ-paris12.fr`

manufacturing technologies and the emergence of just-in-time manufacturing and Kanban control systems, which maintain a limited in-process inventory, the constraint of intermediate buffers with a limited capacity between every two successive machines is introduced. Therefore, the study of flowshop scheduling with limited buffer storage has become attractive for many researchers [28]. In real world applications, the FSP with limited buffers may occur in industrial production, e.g. petrochemical processing industries and cell manufacturing [31], as well as in computer architecture [13]. Papadimitriou and Kanellakis [21] have proved that this problem is strongly NP-hard with respect to the makespan criterion. Therefore, many approximate algorithms have been developed in the literature for solving the FSP with limited buffers.

With the makespan criterion, Dutta and Cunninghan [5] and Reddi [26] have proposed a dynamic programming technique aimed at finding an optimal solution for the problem with two or more machines successively. Leisten [13] has analysed the importance of the limited storage restrictions on the performance of the production systems. Moreover, he has presented several constructive heuristics to solve this variant of FSP. Nowicki [20] has proposed a tabu search approach (TS) by using a non-trivial generalization of the block elimination properties known for the classical FSP. Qian et al. [25] have proposed a differential evolution algorithm (DE) where they developed a rule to convert the continuous version of DE individuals to job permutations. Also, they applied a local search procedure based on the insertion of a part of individuals at each generation. In [31] a genetic algorithm (GA) is adapted to solve the problem where multiple crossover and mutation operators are used simultaneously. Moreover, a decision probability is used to control the utilization of genetic mutation and local search. Liu et al. [14] have presented a particle swarm optimization (PSO) algorithm. They have proposed a new encoding scheme allowing the transition from the continuous values of the PSO to job permutations. A local search procedure and a simulated annealing algorithm (SA) are introduced into the algorithm in order to enrich the search procedure and obtain solutions with good diversity.

In 1996, a new class of evolutionary algorithms was introduced by Mühlenbein and Paaß [19], called Estimation of Distribution Algorithm (EDA). In order to generate a new individual, EDA uses a probabilistic model learned from a population of individuals. Therefore, a distribution of probability is estimated from the selected candidates of the initial population and then new offspring are generated according to this distribution. Finally, according to the quality of the new individual, it is decided whether it will survive in the next generation. Concerning its applications, EDA was devoted to solving several combinatorial optimization problems. A detailed survey of these applications can be found in Larrañaga and Lozano [12], Pelikan [22], and Lozano et al. [15].

In this chapter, we propose an EDA for solving the FSP with limited buffers while minimizing the makespan criterion. Aimed at enhancing the quality of the solution of EDA, it is recommended to use a local search algorithm [15]. The Skewed Variable Neighborhood Search (SVNS) [7] is embedded to the EDA for improving its performance.

2 Problem Definition

In a FSP with limited buffers, there is a set of n jobs to be processed through a set of m machines with the same sequence on all machines and without interruption in the processing (no pre-emption). Let $p_{i[j]}$ be the processing time for the job at position j in the sequence s on the machine i, where $j = 1,2,...,n$, $i = 1,2,...,m$, $s = \{s_1, s_2, ..., s_n\}$ and p_{ij} are fixed, known in advance and have non-negative values. The intermediate storage constraint implies that between each successive pair of machines $(i-1,i)$ a buffer with limited size B_i of unfinished jobs can temporarily stay. Let $D_{i[j]}$ denotes the departure time (starting time) of the job at the j^{th} position in the sequence on the machine i.

The completion time of the last executed job, also known as the makespan $(C_{\max})$ can be found through the recursive expression of the departure time as follows:

$$D_{1[1]} = 0$$
$$D_{1[j]} = D_{1[j-1]} + p_{1[j-1]},\ j = 2,3,...,n$$
$$D_{i[j]} = \max\left(D_{i-1[j]} + p_{i-1[j]}, D_{i[j-1]} + p_{i[j-1]}, D_{i+1[j-B_{i+1}-1]}\right)\ i = 2,...,m, j = 2,...,n$$
Thus,
$$C_{\max} = D_{n[m]} + p_{n[m]}$$

Following this definition, two cases can be discussed. If $B_i = 0$, the limited buffer constraint disappears and the problem will be transformed into the blocking flowshop scheduling problem [28]. Otherwise, if $B_i \geq n-1$, the problem under consideration becomes the classical FSP. Nowicki [20] has presented a detailed analysis of the limited buffer FSP based on a graph representation model.

3 An Overview of Estimation of Distribution Algorithms (EDAs)

The framework of the basic EDA [19] can be presented as follows: starting with a randomly generated initial population, one selects a subpopulation of M parent individuals through a selection method based on the fitness function. Next, one estimates the probability of distribution of the selected parents with a probabilistic model. Then, one generates new offspring, according to the estimated probability distribution. Finally, some individuals in the current population are replaced with newly generated offspring. These steps are repeated until a stopping criterion is met.

Three classes of EDA were developed, according to the chosen probabilistic model. The first class consists of models which do not take into account the dependencies between variables of candidate solutions, i.e. all variables are independent. The second class assumes at most two-order dependencies between these variables and the last class assumes multiple dependencies between the variables.

3.1 EDAs with No Dependencies

3.1.1 Population-Based Incremental Learning Algorithm (PBIL)

The PBIL algorithm was introduced by Baluja [1]. It involves transforming the population of candidate solutions into a probability vector $p(x) = \{p(x_1), p(x_2), ..., p(x_n)\}$ where $p(x_i)$ denotes the probability of a "1" in the i^{th} position of solution bits. In other words, this algorithm checks the proportion of ones in each individual chromosome. All positions are initially equally likely, that is all the probability values are set to 0.5. In this way, the highest level of diversity will be found [1]. In each iteration, after transforming the population into a probability vector, the PBIL algorithm generates P_1 solutions according to the current probability. Then, each component of probability vector, $p(x_i)$, is updated according to the following:

$$p(x_i) = p(x_i) + \alpha(x_i^{best} - p(x_i))$$

where $\alpha \in [0,1]$ is the so-called learning rate and x_i^{best} is the i^{th} bit in the best solution found so far. Using this update rule, the value of $p(x_i)$ increases when x_i^{best} is equal to "1" and decreases otherwise. Furthermore, the values in the probability vector move towards the fittest of the generated solutions. The aim of PBIL is to actively create a probability vector which, with high probability, represents a population of highly evaluated solution vectors. It is worthy to note that each bit is examined independently and thus no interaction is considered.

3.1.2 Compact Genetic Algorithm (CGA)

CGA was proposed by Harik et al. [8]. Similar to PBIL algorithm, CGA creates a probability vector from the population of solutions and all vector values are initialized to 0.5. The update rule used in this algorithm is equivalent to a steady-state tournament selection. So, the procedure is as follows: at each iteration, two individuals are selected at random from the population. These individuals are compared with respect to the fitness values. Then, the probability vector is updated, following this formula:

$$p(x_i) = \begin{cases} p(x_i) + \frac{1}{P} & \text{if } x_i^{best} = 1 \text{ and } x_i^{worst} = 0; \\ p(x_i) - \frac{1}{P} & \text{if } x_i^{best} = 0 \text{ and } x_i^{worst} = 1; \\ p(x_i) & \text{otherwise.} \end{cases}$$

where x_i^{best} and x_i^{worst} denote the i^{th} bit in the best and the worst solutions respectively, among the two selected solutions and P is the population size. According to this method of updating the probability, the CGA replaces one candidate solution by another one using a population of size P.

Two points of difference between PBIL and CGA can be found. Firstly, the update rule used by CGA depends only on the size of the population. Secondly, the CGA is more profitable in terms of memory requirements. The update phase in CGA requires a constant size of $\frac{1}{P}$ whereas the update rule values in PBIL can take an arbitrary value in the range [0,1] and the number of values that can be stored in an element of the vector is infinite.

3.1.3 Univariate Marginal Distribution Algorithm (UMDA)

The UMDA was proposed by Mühlenbein et al. [19]. This algorithm works differently compared to PBIL and CGA. The UMDA does not transform the population into a population vector. After generating a population of individuals, UMDA selects some individuals according to their fitness values. Then, it constructs a probability vector through the selected solution. Formally, the probabilistic model used in UMDA can be written as follows:

$$p(x) = \prod_{i=1}^{n} p(x_i).$$

Finally, it generates new solutions based on the built probabilistic model. The process is repeated until a termination criterion is met. Although UMDA has different framework than CGA and PBIL, the performances of these algorithms are similar [22].

3.2 *EDAs with Two-Order Dependencies*

3.2.1 Mutual Information Maximization for Input Clustering (MIMIC)

MIMIC was proposed by De Bonet et al. [4]. It considers pairwise interactions between the variables in its distribution of probability.

Let $p(x) = p(x_1/x_2,...,x_n)p(x_2/x_3,...,x_n)...p(x_{n-1}/x_n)p(x_n)$ denote the true joint probability, and $\hat{p}_\pi(x) = p(x_{i1}/x_{i2})p(x_{i2}/x_{i3})...p(x_{in-1}/x_{in})p(x_{in})$ where $\pi\{i_1,i_2,...,n\}$ is a permutation of the numbers between 1 and n and $\hat{p}_\pi(x)$ is estimated by the marginal and conditional relative frequencies of the corresponding variables within the subset of selected individuals. The aim of MIMIC algorithm is to generate a permutation π^* that minimizes the divergence between $\hat{p}_\pi(x)$ and $p(x)$. This probabilistic model has the framework of a chain. In a sequential way, all positions, except for the first position of the chain, are conditionally dependent on the previous position in the chain. Starting with an initial random population, the MIMIC algorithm selects a set of promising solutions. After that, it computes the marginal and conditional probabilities. Next, MIMIC uses a greedy algorithm, based on Kullback-Leibler information divergence, to maximize mutual information between the adjacent positions in the chain.

3.2.2 Combining Optimizers with Mutual Information Trees (COMIT)

COMIT algorithm, proposed by Baluja and Davies [2], also considers pairwise interactions between variables of the problem. Instead of using the shape of a chain in the probabilistic model, as in MIMIC, this algorithm attempts to present the probability vector through a tree distribution. The probabilistic model, constructed here, can be written as follows:

$$p(x) = \prod_{i=1}^{n} p(x_i/x_j).$$

where i is a descendant node from j. If i is a root node then $p(x_i/x_j) = p(x_i)$. The objective is to find a tree that maximizes mutual information between parent nodes and their descendants. In [2], it is shown that COMIT outperforms MIMIC algorithm.

3.2.3 Bivariate Marginal Distribution Algorithm (BMDA)

BMDA was introduced by Pelikan and Mühlenbein [24]. It constitutes a kind of a generalization of both the previous algorithms and UMDA. For constructing the probability distribution, this algorithm uses a set of disjoint tree distributions. The criterion used for measuring the dependencies between variables is Pearson's chi-square statistics using the following:

$$\chi^2_{ij} = M \times \sum_{x_i, x_j} \frac{(p(x_i, x_j) - p(x_i)p(x_j))^2}{p(x_i)p(x_j)}$$

3.3 EDAs with Multiple Dependencies

3.3.1 Factorized Distribution Algorithm (FDA)

FDA was proposed by Mühlenbein and Mahnig [17]. The method involves using an additive decomposition of function (ADF), to decompose the fitness function of the problem and a factorization method, based on Boltzmann distribution which is used to factor the distribution into consistent, conditional and marginal distributions. In such a way, N individuals are generated according to uniform distribution as in UMDA. Then, the Boltzmann selection is used to select M individuals from those generated previously. In addition, the conditional probabilities are estimated from the selected individuals. Finally, M new individuals are generated according to these probabilities. The process is repeated until a termination criterion is met.

3.3.2 The Extended Compact Genetic Algorithm (ECGA)

The ECGA [8] is an extension inspired from CGA. It works by partitioning the variables into different partitions by using the so called Marginal Product Model (MPM). This model does not estimate the conditional probabilities as in FDA. However, it includes both univariate marginal distribution as well as multivariate marginal distribution. The form of probability distribution used in MPM can be presented as:

$$p(x) = \sum_{i \in \Omega} p(x_i)$$

where Ω is the set of alleles included in the same partitions. Based on MPM, ECGA uses a greedy procedure in order to obtain the best partitions. At the beginning, MPM assumes that all variables are independent, i.e. all partitions are composed of one element. Then, at each iteration, it merges two partitions together with respect to the Bayesian Information criterion. Finally, when no further possible improvement is obtained, the model used is retained.

3.3.3 Bayesian Optimization Algorithm (BOA)

In [23], the BOA was developed. This algorithm considers multiple interactions among the variables. At first, an initial random population is generated. Secondly, a selected method is chosen to select the promising solutions. Thirdly, a Bayesian Network is constructed from a probabilistic model based on the selected individuals. Formally, an acyclic Bayesian network with directed edges is used in this algorithm. The estimated joint probability can be written as follows:

$$p(x) = \prod_{i=0}^{n-1} p(x_i / \Phi_{x_i})$$

where Φ_{x_i} is the set of nodes from which there exists an edge to x_i. In such way, a variable is sampled after all its parents have already been sampled. In order to evaluate the quality of the constructed network, a metric named Bayesian Dirichlet metric is used by Pelikan et al. [23]. Finally, new offspring are generated according to this network.

3.3.4 Learning Factorised Distribution Algorithm (LFDA)

LFDA [18] extends the FDA by using the Bayesian network. Instead of using the ADF structure, as in FDA, LFDA uses the Bayesian network framework to build the probabilistic model. In fact, this algorithm does not need a priori information about the structure of the problem. Therefore, LFDA performs similarly to BOA with the exception of the metric used . In this context, LFDA uses a Bayesian Information Criterion (BIC) in order to assess the network's quality.

3.3.5 Estimation of Bayesian Network Algorithm (EBNA)

EBNA was proposed by Etxeberria and Larrañaga [6]. Also, these authors exploit the Bayesian network structure in estimating the joint probability. This algorithm follows the same process as BOA and LFDA. In [11] three different variants of EBNA have been proposed where they differ in the metrics used in the evaluation step. The first one, called EBNA$_{PC}$, uses Chi square tests to detect the conditional independencies. The BIC criterion is devoted to the second variant EBNA$_{BIC}$. The last one is called EBNA$_{k_2+pen}$ which uses the marginal likelihood in order to construct the Bayesian network and a penalty is added to the metric for reducing the complexity of the algorithm.

4 Estimation of Distribution Algorithm (EDA) for the Flowshop Scheduling Problem with Limited Buffer

EDA is a new evolutionary algorithm proposed by Mühlenbein and Paaß in 1996 [19]. Instead of recombination and mutation, EDA generates new individuals with respect to a probabilistic model, learned from the population of parents. To the best of our knowledge, in the literature, there is no application of this algorithm in the scheduling context, with the exception of the work of Jarboui et al. [10]. In this section, we discuss, in detail, our proposed EDA for solving the FSP with respect to makespan criterion.

4.1 Encoding Solutions and Initial Population

For encoding the solution, we use the well-known representation scheme for the permutation FSP, which is the permutation of n jobs, where the j^{th} number in the permutation denotes the job located at position j. For the purpose of diversity, the initial population of P individuals is generated randomly.

4.2 Selection

The selection procedure adopted in our algorithm consists of two phases. First, the individuals of the initial population are sorted according to their objective function value. Second, M individuals are selected from the subset of 20% of the best individuals from the sorted list.

4.3 Probabilistic Model: Construction of New Solution

The choice of the probabilistic model is closely related to the performance of the EDA [15]. So, it is recommended to carefully construct this model. In this chapter, the probabilistic model is built while taking into account the specific characteristics of the problem under consideration, as in [10]. Therefore, both the order of jobs in the sequence and the similar blocks of jobs presented in the selected parents are considered. In order to construct a new sequence, Jarboui et al. [10] have proposed a probability π_{jk} for selecting a job j at position k:

$$\pi_{jk} = \frac{\eta_{jk} \times \mu_{j[k-1]}}{\sum_{l \in \Omega_k} \eta_{lk} \times \mu_{l[k-1]}}$$

with η_{jk} denotes the number of times job j that appears before or at the position k in the subset of the selected sequences plus a given constant δ_1. This parameter indicates the importance of the order of jobs in the sequence. $\mu_{j[k-1]}$ denotes the

EDA Algorithm

```
Initialize the population $P$ using random permutation;
loop
    select $M$ individuals from 20% of the best individuals in the population;
    select $s_0$ at random from the $M$ selected individuals;
    for $k = 1, 2, \ldots, n$
        update $\Omega_k$ using the $q$ first job not already scheduled from $s_0$;
        compute the probability $\pi_{jk}$ for each job $j \in \Omega_k$;
        $s_{current,k}$ receives the job selected from $\Omega_k$ following the computed probability;
    end for
    if (the fitness of newly created induvidual $s_{current}$ is better than the worst individual in the
        population) then
        replace the latter by $s_{current}$;
    end if
until (stopping criterion is reached)
```

Fig. 1 EDA procedure

number of times job j comes immediately after the job at the position $k-1$ in the new individual from the sequences of the selected parents. This parameter is used to highlight the importance of similar blocks in the sequence. Ω_k represents the set of jobs not already scheduled until position k. According to this model, the task of constructing new offspring has a complexity order of $O(n^2)$. Although this model provides a good solution quality, it consumes high CPU times, especially when the size of the problem increases. In this work, we propose an improvement to the model of Jarboui et al. [10] by reducing its complexity. First, a sequence of jobs s_0 is selected, at random, among the M selected parents. Then, the probabilistic model is built like the one described above while modifying the set Ω_k into Ω_{kq}. The latter set is composed of q jobs not already scheduled, following their order in s_0, until position k. So, while comparing the model proposed here and the one proposed by Jarboui al. [10], we find that the phase of construction of new individual requires a linear time $O(n)$ in this proposed model. Moreover, this formulation grants more intensification to the algorithm. The pseudo-code of the EDA is given in Fig. 1.

4.4 Skewed Variable Neighborhood Search (SVNS) Algorithm

SVNS algorithm is an extension inspired by the original variable neighborhood search algorithm proposed by Hansen and Mladenovìc [7]. In order to escape from the local optima, this algorithm is added to our EDA after creating a new solution. Firstly, to increase the performance of the SVNS, we restrict the application of the algorithm to only a part of the individuals [29]. To do this, we define a calculated probability p^c, as in [10], which decides if the new individual will be subject to an application of the SVNS algorithm. This probability depends on the quality of the created individual, as follows:

$$p^c = \max\left\{\exp\left(\frac{RD}{\alpha}\right), \varepsilon\right\},$$

where

$$RD = \left(\frac{f(s_{current}) - f(s_{best})}{f(s_{best})}\right)$$

and $s_{current}$ and s_{best} denote the sequences of the new offspring and the best individual found so far, respectively. This probability depends on the distance separating the current solution and the best solution. Therefore, the best obtained solutions have a high probability of being subject to an application of SVNS algorithm.

A random number in the range $[0,1]$ is drawn for each individual. If this number is less than or equal to p^c, then the considered individual will be eligible to be subjected to the SVNS algorithm. This algorithm consists of two structures of neighbourhood $(k = 1,2)$. The first structure leads to the selection of two distinct positions at random from $s_{current}$ and the permutation of the jobs located on those positions. The second one consists of selecting, randomly, a job j from $s_{current}$ and inserting it at a randomly selected position. After setting k to 1, the shaking phase generates a sequence s_1 according to the k^{th} neighborhood. This phase is introduced into the SVNS algorithm in order to avoid cycling which might occur if any deterministic rule was used.

Then, two local search methods are employed. The first procedure performs all possible swap moves in the sequence s_1. The obtained local optimum is denoted by s_2 (see Fig. 2). After that, the second local search procedure starts performing all insertion moves to s_2. Let s_3 be the resulting local optimum (see Fig. 3). If $f(s_3) < f(s_2)$, we replace s_1 by s_3, and we return to the first local search procedure. The process is repeated until no improvement is possible. Then we compare the objective

Swap Local Search

```
swap_local_search()
Let $s_0$ be the input solution
Let $s_{best}$ be the output solution
$s_{best} = s_0$;
set $i = 1$;
loop
    for $j = i+1, i+2, ..., n$
        create a new solution by exchanging the jobs at position $i$ and $j$ in the best sequence;
        if  (the fitness of the new solution is better than the $s_{best}$) then
            update $s_{best}$;
            break;
        end if
    endfor
    $i = i+1$;
    if $(i > n-1)$  set $i = 1$;
until (no possible improvement)
```

Fig. 2 Swap local search procedure

function values of s_3 (the local optimum) and $s_{current}$ (the incumbent solution). If s_3 is better than $s_{current}$, then the latter is updated and we go to the shaking phase with $k = 1$. Else, we define a distance function $\rho(s_3, s_{current})$ as a discrete function, as follows:

$$\rho(s_3, s_{current}) = \frac{\sum_{j=1}^{n} \left| pos_3^j - pos_{current}^j \right|}{n}$$

where pos_3^j denotes the position of the job j in s_3 and $pos_{current}^j$ denotes the position of the job j in the incumbent sequence $s_{current}$. So, $\rho(s_3, s^{current})$ represents the position lags between the obtained local optimum and the incumbent solution. The calculation of this distance function is illustrated by the example below. Let $s_3 = \{6,4,3,1,2,5\}$ and $s_{current} = \{3,1,6,4,2,5\}$, we define by pos_3 and $pos_{current}$ two n-dimensional vectors, where pos_3^j and $pos_{current}^j$ indicate the positions of the job j in s_3 and $s_{current}$, respectively. So, $pos_3 = \{4,5,3,2,6,1\}$ and $pos_{current} = \{2,5,1,4,6,3\}$ and consequently:

$$\rho(s_3, s^{current}) = \frac{|4-2| + |5-5| + |3-1| + |2-4| + |6-6| + |1-3|}{6} = 1.33$$

Also, we define a parameter β: when $f(s_3) - f(s_{current}) < \beta \times \rho(s_3, s_{current})$, then we accept the deterioration of the current solution and we replace $s_{current}$ by s_3 and thereafter k is reinitialized to 1. Otherwise, k is incremented to 2 and we return to the shaking process. Finally, the whole procedure is repeated until a maximum number of iterations ($iter_{\max}$) is reached. Fig. 4 describes the pseudo-code of the SVNS algorithm.

```
Insert Local Search
Let $s_0$ be the input solution
Let $s_{best}$ be the output solution
$s_{best} = s_0$;
set $i = 1$;
loop
    for $j = 1,2,...,n$ and $j \neq i$
        create a new solution by inserting the job at position $i$ into position $j$ in the best sequence;
        if  (the fitness of the new solution is better than the $s_{best}$) then
            update $s_{best}$;
            break;
        end if
    endfor
    $i = i + 1$;
    if $(i > n)$  set $i = 1$;
until (no possible improvement)
```

Fig. 3 Insert local search procedure

SVNS Algorithm

```
Let $s_0$ be the input solution
Let $s_{best}$ be the output solution
$s_{best} = s_0$;
$s_{current} = s_0$;
loop
    set $k = 1$;
    do
        $s_1 = s_{current}$;
        if $(k = 1)$ then
            select $i$ and $j$ at random and insert the job at position $i$ into position $j$ in $s_1$;
        else
            select $i$ and $j$ at random and exchange the jobs at position $i$ and $j$ in $s_1$;
        endif
        do
            find the optimum local $s_2$ by applying swap local search to $s_1$;
            find the optimum local $s_3$ by applying insert local search to $s_2$;
            if ($s_3$ is better than $s_2$) then
                $s_1 = s_3$;
            endif
        while ($s_3$ is better than $s_2$)
        if ($s_3$ is better than $s_{best}$) then
            $s_{best} = s_3$;
            $s_{current} = s_3$;
            $k = 1$;
        else if $(f(s_3) - f(s_{current}) < \beta \times \rho(s_3, s_{current}))$ then
            $s_{current} = s_3$;
            $k = 1$;
        else
            $k = k + 1$;
        endif
    while$(k \leq 2)$
until(stopping criterion is reached)
```

Fig. 4 SVNS procedure

4.5 Replacement and Stopping Criterion

In our algorithm, we compare the new individual with the worst individual in the current population. If the first one is better than the latter and the sequence of the offspring is unique, then the worst individual is removed from the population and is replaced by the new individual. In the literature, various stopping criteria have been proposed, such as the maximum number of generations, bound of time, the maximum number of iterations without improvement, etc. In our work, we set a maximal computational time. The pseudo-code of the whole hybrid algorithm is provided in Fig. 5.

```
Hybrid EDA
Initialize the population $P$ using random permutation;
loop
    select $M$ individuals from 20% of best individuals in the population;
    select $s_0$ at random from the $M$ selected individuals;
    for $k = 1,...,n$
        update $\Omega_k$ using the $q$ first job not already scheduled from $s_0$;
        compute the probability $\pi_{jk}$ for each job $j \in \Omega_k$;
        $s_{current,k}$ receives the job selected from $\Omega_k$ following the computed probability;
    end for
    compute $p^c$;
    if ($rand \leq p^c$) then // rand denotes a random value uniformly distributed in the range $[0,1]$
        Apply the SVNS procedure to improve $s_{current}$;
    end if
    if (the fitness of new created induvidual $s_{current}$ is better than the worst individual in the
        population) then
        replace the latter by $s_{current}$;
    end if
until (stopping criterion is reached)
```

Fig. 5 EDA with SVNS procedure

5 Experimental Results and Comparative Study

The algorithm was coded in C++ programming language. All experiments with EDA for FSP with limited buffers, with respect to the makespan criterion, were run in Windows XP on a desktop PC with Intel Pentium IV, 3.2 GHz processor and 512 MB memory. EDA was tested for 29 instances distributed as follows: 8 instances of Carlier [3] and 21 instances of Reeves [27]. All results were obtained after 20 replications for each instance.

The corresponding parameters of EDA were set, experimentally, to the following values: $P = 20$, $\delta_1 = \delta_2 = 4/n$, the number of the selected parents $M = 3$, the number of generated offspring $O = 3$. Numerically, $p^c = 0.5$ leads to the acceptance of a sequence with a makespan relatively superior by 1% to the best value of the makespan found so far. So, thereafter we determined p^c according to this formula: $p^c = \max\left\{\exp\left(\frac{RD}{\alpha}\right), \varepsilon\right\}$, with $\varepsilon = 0.01$. The maximum number of iterations of VNS algorithm $(iter_{\max})$ was set at 3 and the coefficient β was fixed to 2. The performance measure employed in our numerical study was the average relative percentage deviation in the makespan:

$$\Delta_{avg} = \frac{\sum_{i=1}^{R}\left(\frac{Heu_i - LB}{LB}\right)}{R} \times 100,$$

Heu_i is the objective value of the solution found by our algorithm at the i^{th} replication and LB (Lower Bound) is the best known value of makespan when the size of the buffer is infinite.

The comparative approaches used in this study consist of the representative methods employed for solving this problem in the literature, including Hybrid Genetic Algorithm (HGA) of Wang et al. [31], Hybrid Particle Swarm Optimization (HPSO) of Liu et al. [14] and (PSOvns) of Tasgetiren al. [30] .

5.1 Results of the Proposed EDA under Zero and Infinite Buffer Sizes

Table 1 and Table 2 summarize the results obtained by our EDA against HGA and HPSO for the extreme cases of the problem. In Table 1, the problem with no buffer,

Table 1 Computational results under $B_i = 0$

		HGA		HPSO		EDA	
instances	*LB*	$\Delta_{\min}$	Δ_{avg}	$\Delta_{\min}$	Δ_{avg}	$\Delta_{\min}$	Δ_{avg}
Car1	7038	5.27	6.56	5.27	6.46	5.27	5.27
Car2	7166	8.39	10.2	5.14	7.75	4.70	4.70
Car3	7312	10.2	11.06	9.6	10.02	9.60	9.60
Car4	8003	10.41	12.44	9.18	11.18	9.18	9.18
Car5	7720	6.45	7.19	6.45	7.51	6.45	6.45
Car6	8505	4.33	4.71	4.33	4.64	4.33	4.33
Car7	6590	3.00	3.00	3.01	3.01	3.00	3.00
Car8	8366	2.62	3.41	2.62	3.18	2.62	2.62
Rec1	1247	15.8	17.8	13.87	15.77	13.71	13.76
Rec3	1109	15.78	17.77	14.43	15.95	12.98	13.66
Rec5	1242	16.59	18.39	14.9	16.52	14.09	14.13
Rec7	1566	13.22	14.26	12.13	13.28	10.34	10.51
Rec9	1537	13.27	14.53	12.17	13.59	11.19	11.23
Rec11	1431	12.23	13.73	11.39	12.49	10.97	11.02
Rec13	1930	9.64	10.24	9.48	10.32	9.02	9.20
Rec15	1950	7.69	9.62	7.03	8.85	6.41	6.52
Rec17	1902	10.04	11.52	9.46	10.88	9.46	9.46
Rec19	2093	16.96	18.11	14.96	16.74	14.48	15.12
Rec21	2017	19.53	20.67	17.8	18.94	16.51	16.99
Rec23	2011	16.61	18.34	15.12	17.12	13.77	14.37
Rec25	2513	15.68	17.6	13.33	14.77	11.98	12.46
Rec27	2373	14.83	15.75	14.08	15.18	12.85	13.78
Rec29	2287	13.29	14.21	13.21	13.9	12.46	13.15
Rec31	3045	24.6	26.49	20.99	22.83	20.36	21.18
Rec33	3114	25.56	26.4	22.19	23.34	20.84	21.66
Rec35	3277	18.71	19.66	15.17	16.39	13.70	14.74
Rec37	4951	25.77	26.87	23.77	24.64	23.13	24.55
Rec39	5087	25.56	26.5	22.33	23.35	21.37	22.39
Rec41	4960	29.01	30.02	26.27	27.42	25.95	26.38
average		14.17	15.42	12.75	14.00	12.09	12.46

Table 2 Computational results under B_i = infinite

		HGA		HPSO		EDA	
instances	*LB*	$\Delta_{\min}$	Δ_{avg}	$\Delta_{\min}$	Δ_{avg}	$\Delta_{\min}$	Δ_{avg}
Car1	7038	0.00	0.00	0.00	0.00	0.00	0.00
Car2	7166	0.00	0.59	0.00	0.00	0.00	0.00
Car3	7312	0.00	0.86	0.00	0.79	0.00	0.00
Car4	8003	0.00	0.00	0.00	0.00	0.00	0.00
Car5	7720	0.00	0.00	0.00	0.59	0.00	0.00
Car6	8505	0.00	0.57	0.00	0.61	0.00	0.00
Car7	6590	0.00	0.00	0.00	0.00	0.00	0.00
Car8	8366	0.00	0.26	0.00	0.03	0.00	0.00
Rec1	1247	0.00	1.36	0.00	0.41	0.00	0.04
Rec3	1109	0.00	1.35	0.18	0.30	0.00	0.00
Rec5	1242	0.00	0.40	0.24	0.29	0.24	0.24
Rec7	1566	0.00	1.91	0.70	1.66	0.00	0.00
Rec9	1537	0.78	2.26	0.00	1.54	0.00	0.00
Rec11	1431	0.00	3.09	0.00	1.20	0.00	0.00
Rec13	1930	0.73	2.08	0.21	1.25	0.00	0.14
Rec15	1950	0.82	1.66	0.67	1.36	0.00	0.24
Rec17	1902	1.37	3.36	0.00	2.33	0.00	0.06
Rec19	2093	1.29	2.85	0.67	1.35	0.29	0.51
Rec21	2017	1.44	2.50	1.44	1.61	1.44	1.44
Rec23	2011	1.39	3.47	0.90	1.84	0.45	0.52
Rec25	2513	2.51	3.69	1.11	2.42	0.52	0.94
Rec27	2373	1.39	2.80	0.55	1.83	0.42	0.99
Rec29	2287	3.10	3.92	1.01	3.05	0.26	0.89
Rec31	3045	3.19	3.88	1.38	2.34	0.49	1.67
Rec33	3114	0.83	2.08	0.00	0.78	0.13	0.64
Rec35	3277	0.00	0.21	0.00	0.01	0.00	0.00
Rec37	4951	3.80	4.77	2.26	3.03	1.66	2.99
Rec39	5087	2.95	3.62	1.47	2.11	1.22	1.88
Rec41	4960	4.92	5.53	2.74	3.48	2.42	3.24
average		1.05	2.04	0.54	1.25	0.33	0.57

i.e. the blocking flowshop scheduling problem, is addressed. It is clear that EDA is better than HGA and HPSO with respect to both δ_{avg} and the minimum relative percentage deviations of best found solutions, $\delta_{\min}$. Therefore, we observe that, in average for all instances, δ_{avg} provided by our algorithm (12,46 %) is less than $\delta_{\min}$ provided by the compared algorithms (14,17% for HGA and 12,75% for HPSO). When the size of the buffer is infinite, i.e. for the classical FSP, the EDA again outperforms the other approaches in terms of δ_{avg} and $\delta_{\min}$ (Table 2). Moreover, for Carlier instances [3], our EDA seems to be able to reach all *LB* values for all replications in short CPU times (Table 6). Globally, the EDA reaches 17 *LB* values out of 29 and 14 *LB* values are obtained by HGA and HPSO.

Table 3 Computational results under $B_i = 1$

		HGA		PSOvns		HPSO		EDA	
instances	*LB*	$\Delta_{\min}$	Δ_{avg}	$\Delta_{\min}$	Δ_{avg}	$\Delta_{\min}$	Δ_{avg}	$\Delta_{\min}$	Δ_{avg}
Car1	7038	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Car2	7166	0.00	0.88	0.00	0.15	0.00	0.00	0.00	0.00
Car3	7312	0.00	1.05	0.00	0.84	0.00	0.91	0.00	0.00
Car4	8003	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00
Car5	7720	0.00	0.17	0.00	0.23	0.00	0.22	0.00	0.00
Car6	8505	0.00	0.57	0.00	0.27	0.00	0.50	0.00	0.00
Car7	6590	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00
Car8	8366	0.00	0.78	0.00	0.03	0.00	0.00	0.00	0.00
Rec1	1247	1.84	3.34	0.96	2.13	0.32	1.04	0.16	0.18
Rec3	1109	1.98	2.88	1.44	2.99	0.54	1.84	0.18	0.25
Rec5	1242	0.56	1.38	1.45	2.37	0.40	1.39	0.24	0.30
Rec7	1566	1.79	3.51	1.15	1.76	0.13	2.26	0.00	0.11
Rec9	1537	1.76	2.96	1.63	2.62	0.98	1.86	0.33	0.46
Rec11	1431	2.24	4.97	1.33	2.53	0.14	1.65	0.14	0.19
Rec13	1930	1.97	2.50	1.40	2.41	0.78	1.67	0.36	0.58
Rec15	1950	1.18	1.64	1.13	2.29	1.08	1.45	0.00	0.34
Rec17	1902	2.52	4.28	2.79	3.70	1.05	2.38	0.89	0.89
Rec19	2093	3.11	4.27	3.20	3.96	2.10	2.88	1.15	1.55
Rec21	2017	2.88	4.59	2.83	4.34	1.69	3.16	1.64	1.85
Rec23	2011	4.92	6.63	3.33	4.40	2.54	3.88	1.49	2.04
Rec25	2513	3.90	5.48	3.42	4.88	2.71	3.89	1.63	2.27
Rec27	2373	3.50	4.66	3.88	4.54	2.40	3.39	1.69	2.01
Rec29	2287	3.76	3.99	3.41	5.45	2.93	3.69	1.75	2.28
Rec31	3045	6.37	7.28	4.99	5.90	3.55	4.75	2.50	3.38
Rec33	3114	4.34	5.42	3.82	4.98	2.99	3.80	1.86	2.68
Rec35	3277	1.77	2.82	0.76	1.42	0.00	0.45	0.00	0.09
Rec37	4951	8.14	8.73	8.14	8.97	6.14	7.18	5.82	6.70
Rec39	5087	6.17	6.84	5.98	6.80	4.40	5.42	3.87	4.57
Rec41	4960	8.06	8.74	7.92	9.00	6.25	7.03	6.03	6.60
average		2.51	3.46	2.24	3.07	1.49	2.30	1.09	1.36

5.2 Results of the Proposed EDA under Different Buffer Sizes

Even for the buffer sizes of 1, 2 and 4 where the results are depicted in Tables 3, 4 and 5 respectively, our proposed algorithm remains the best one in terms of the quality of the solutions. Moreover, it is shown that while the buffer sizes increase the deviations become close to zero. Therefore, the performance of the EDA increases when the problem presents fewer constraints and approaches the classical FSP. Also, it can be seen that the differences between $\delta_{\min}$ and δ_{avg} are very small, which implies the high robustness of our algorithm in a consistent manner (Table 6).

Table 4 Computational results under $B_i = 2$

		HGA		PSOvns		HPSO		EDA	
instances	*LB*	$\Delta_{\min}$	Δ_{avg}	$\Delta_{\min}$	Δ_{avg}	$\Delta_{\min}$	Δ_{avg}	$\Delta_{\min}$	Δ_{avg}
Car1	7038	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Car2	7166	0.00	0.59	0.00	0.15	0.00	0.00	0.00	0.00
Car3	7312	0.00	0.93	0.00	0.89	0.00	0.68	0.00	0.00
Car4	8003	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00
Car5	7720	0.00	0.04	0.00	0.35	0.00	0.66	0.00	0.00
Car6	8505	0.00	0.53	0.00	0.27	0.00	0.69	0.00	0.00
Car7	6590	0.00	0.00	0.00	0.35	0.00	0.00	0.00	0.00
Car8	8366	0.00	0.52	0.00	0.03	0.00	0.03	0.00	0.00
Rec1	1247	0.00	1.32	0.16	0.81	0.16	0.46	0.00	0.05
Rec3	1109	0.72	1.74	0.18	0.58	0.00	0.32	0.00	0.00
Rec5	1242	0.24	0.51	0.24	0.87	0.24	0.35	0.24	0.24
Rec7	1566	1.15	2.69	1.15	1.45	0.00	1.90	0.00	0.00
Rec9	1537	1.43	2.37	0.00	1.59	0.00	1.43	0.00	0.00
Rec11	1431	0.28	3.13	0.14	2.28	0.00	1.45	0.00	0.00
Rec13	1930	1.35	2.22	1.24	2.44	0.10	1.24	0.00	0.21
Rec15	1950	1.18	1.72	0.97	1.97	0.56	1.48	0.05	0.31
Rec17	1902	2.37	4.21	1.05	3.12	0.42	3.18	0.00	0.02
Rec19	2093	1.58	3.07	1.48	2.71	0.62	1.43	0.29	0.69
Rec21	2017	2.53	3.03	1.64	2.22	1.64	1.70	0.59	1.39
Rec23	2011	2.88	4.06	1.39	2.80	0.85	1.90	0.40	0.52
Rec25	2513	2.83	3.73	2.71	3.61	0.80	2.38	0.76	1.21
Rec27	2373	1.81	2.97	2.15	3.39	1.56	2.35	0.55	1.07
Rec29	2287	3.85	3.97	3.24	4.37	1.27	2.58	0.79	1.07
Rec31	3045	2.66	3.44	3.02	3.96	1.61	2.43	0.53	1.69
Rec33	3114	1.73	2.63	1.32	1.91	0.64	1.17	0.22	0.71
Rec35	3277	0.00	0.76	0.00	0.55	0.00	0.05	0.00	0.00
Rec37	4951	4.87	5.88	4.93	6.01	2.97	3.79	2.54	3.52
Rec39	5087	4.42	5.40	3.54	4.22	1.71	2.69	1.30	2.13
Rec41	4960	5.34	5.90	5.18	5.74	3.19	3.76	2.72	3.73
average		1.49	2.32	1.23	2.02	0.63	1.38	0.38	0.64

5.3 Comparative Results Based on Unilateral Paired t-Test

In order to compare the performance of our algorithms, we have used the unilateral paired t-test procedure [16] at the 99% significance level. This procedure consists of testing the differences in two samples of observations within pairs. Let μ_A and μ_B denote the true average relative percent deviations of the algorithms A and B respectively. The tested hypotheses are:

$$\begin{cases} H_0 : \mu_A - \mu_B = 0 \\ H_1 : \mu_A - \mu_B < 0 \end{cases}$$

Table 5 Computational results under $B_i = 4$

		HGA		PSOvns		HPSO		EDA	
instances	*LB*	$\Delta_{\min}$	Δ_{avg}	$\Delta_{\min}$	Δ_{avg}	$\Delta_{\min}$	Δ_{avg}	$\Delta_{\min}$	Δ_{avg}
Car1	7038	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Car2	7166	0.00	0.89	0.00	0.15	0.00	0.00	0.00	0.00
Car3	7312	0.00	1.11	0.00	0.55	0.00	1.00	0.00	0.00
Car4	8003	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Car5	7720	0.00	0.19	0.00	0.47	0.00	0.85	0.00	0.00
Car6	8505	0.00	0.50	0.00	0.27	0.00	0.42	0.00	0.00
Car7	6590	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00
Car8	8366	0.00	0.48	0.00	0.07	0.00	0.09	0.00	0.00
Rec1	1247	0.00	1.53	0.16	0.54	0.16	0.42	0.00	0.05
Rec3	1109	0.18	1.36	0.09	0.50	0.00	0.36	0.00	0.00
Rec5	1242	0.24	0.60	0.24	0.80	0.24	0.38	0.24	0.24
Rec7	1566	0.00	2.17	0.89	1.60	0.38	1.87	0.00	0.00
Rec9	1537	0.98	2.26	0.39	1.55	0.00	1.40	0.00	0.00
Rec11	1431	0.00	3.08	0.49	2.25	0.00	0.82	0.00	0.00
Rec13	1930	1.14	2.15	0.99	2.43	0.26	1.45	0.00	0.11
Rec15	1950	1.18	1.65	0.92	2.39	0.72	1.67	0.00	0.23
Rec17	1902	2.58	4.26	1.84	3.00	0.00	2.43	0.00	0.06
Rec19	2093	1.29	2.96	1.96	2.73	0.67	1.58	0.29	0.56
Rec21	2017	1.64	2.42	1.49	2.18	1.44	1.62	0.55	1.38
Rec23	2011	1.39	3.69	0.95	2.42	0.60	2.14	0.45	0.51
Rec25	2513	2.43	3.46	2.47	3.51	0.72	2.25	0.48	0.97
Rec27	2373	1.98	3.02	1.43	2.89	0.97	1.84	0.72	0.96
Rec29	2287	2.62	3.90	2.36	4.40	1.05	3.08	0.48	0.86
Rec31	3045	3.02	3.71	2.96	3.79	1.18	2.21	0.99	1.74
Rec33	3114	0.77	1.94	0.93	1.63	0.35	0.83	0.00	0.64
Rec35	3277	0.00	0.29	0.00	0.24	0.00	0.01	0.00	0.00
Rec37	4951	4.20	5.01	4.83	5.44	2.65	3.22	2.18	3.16
Rec39	5087	3.09	3.65	2.99	4.01	1.34	2.24	1.14	1.85
Rec41	4960	4.82	5.53	4.84	5.31	2.70	3.46	2.20	3.36
average		1.16	2.13	1.15	1.90	0.53	1.30	0.33	0.58

H_0 implies that the average relative percent deviations of the two algorithms are similar, while H_1 implies that the average relative percent deviations of the algorithm A are lower than the ones for the algorithm B. The obtained results are given in Table 7.

For different buffer sizes, the statistical tests prove that the negative difference between EDA and all other algorithms is meaningful at the significance level of 0.99. Therefore, the effectiveness of our algorithm against the representative approaches from the literature for this problem is proved.

Table 6 Average CPU times of EDA over 20 replications (in seconds)

instances	buffer 0	buffer 1	buffer 2	buffer 4	infinite
car1	0.03	0.00	0.00	0.00	0.00
car2	0.01	0.00	0.00	0.00	0.00
car3	0.00	0.01	0.01	0.02	0.01
car4	0.05	0.00	0.00	0.00	0.00
car5	0.00	0.00	0.02	0.02	0.02
car6	0.00	0.00	0.00	0.00	0.00
car7	0.00	0.00	0.00	0.00	0.00
car8	0.00	0.00	0.00	0.00	0.00
Rec1	0.61	0.55	0.38	0.41	0.29
Rec3	0.46	0.60	0.25	0.26	0.24
Rec5	0.55	0.51	0.01	0.01	0.01
Rec7	0.97	1.43	0.63	0.75	0.67
Rec9	0.98	1.33	0.27	0.17	0.17
Rec11	0.94	0.91	0.15	0.14	0.16
Rec13	1.83	2.01	1.85	1.66	1.51
Rec15	1.90	1.93	2.17	1.95	2.39
Rec17	1.53	1.04	1.34	1.77	1.29
Rec19	2.03	2.04	2.10	2.07	2.10
Rec21	2.09	2.27	1.50	0.72	0.59
Rec23	2.77	2.52	1.60	2.23	2.48
Rec25	3.29	2.66	3.36	2.59	3.41
Rec27	2.88	3.15	3.39	2.01	2.98
Rec29	3.61	3.64	3.03	2.95	3.14
Rec31	3.90	4.01	3.85	3.71	4.01
Rec33	3.72	4.06	3.54	2.21	2.06
Rec35	4.38	3.37	1.20	0.78	0.79
Rec37	16.45	15.00	15.96	14.49	14.96
Rec39	15.92	15.27	16.19	14.03	15.63
Rec41	14.10	15.46	13.87	14.47	15.92
average	2.93	2.89	2.64	2.39	2.58

Table 7 Comparative results based on unilateral paired t-test

H_0	H_1	t-value	p-value
	B_i =infinite		
EDA=HGA	EDA<HGA	-7.5116	0.000
EDA=HPSO	EDA<HPSO	-5.4095	0.000
	$B_i = 0$		
EDA=HGA	EDA<HGA	-9.9352	0.000
EDA=HPSO	EDA<HPSO	-10.0478	0.000
	$B_i = 1$		
EDA=HGA	EDA<HGA	-8.4649	0.000
EDA=HPSO	EDA<HPSO	-8.1492	0.000
EDA=PSOvns	EDA<PSOvns	-9.0326	0.000
	$B_i = 2$		
EDA=HGA	EDA<HGA	-7.7635	0.000
EDA=HPSO	EDA<HPSO	-5.4987	0.000
EDA=PSOvns	EDA<PSOvns	-7.4854	0.000
	$B_i = 3$		
EDA=HGA	EDA<HGA	-7.6886	0.000
EDA=HPSO	EDA<HPSO	-5.4696	0.000
EDA=PSOvns	EDA<PSOvns	-6.9382	0.000

6 Conclusion

This chapter presented an application of EDA to a FSP under limited buffer with respect to makespan criterion. Therefore, we developed a probabilistic model based on the main characteristics of the problem. After constructing new sequence of jobs through the EDA, a SVNS was added to improve the quality of the solution. The computational results show that our algorithm outperforms the approaches it was compared with in terms of the solution quality and gives good results in short CPU times. Moreover, the performance of our algorithm is proved by using the unilateral paired t-test procedure.

References

1. Baluja, S.: Populationbased incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report N°CMU-CS-94-163. Carnegie Mellon University, Pittsburgh (1994)
2. Baluja, S., Davies, S.: Using optimal dependencytrees for combinatorial optimization: Learning the structure of the search space. In: Proceedings of the 14th International Conference on Machine Learning, pp. 30–38. Morgan Kaufmann, San Francisco (1997)
3. Carlier, J.: Ordonnancements a contraintes disjonctives. RAIRO-Operations Research 12, 333–351 (1978)
4. De Bonet, J.S., Isbell, C.L., Viola, P.: MIMIC: Finding optima by estimating probability densities. In: Mozer, M.C., Jordan, M.I., Petsche, T. (eds.) Advances in Neural Information Processing Systems, vol. 9, pp. 424–431. The MIT Press, Cambridge (1997)

5. Dutta, S.K., Cunningham, A.A.: Sequencing two-machine flow-shops with finite intermediate storage. Management Science 21, 989–996 (1975)
6. Etxeberria, R., Larrañaga, P.: Global optimization with Bayesian networks. In: II Symposium on Artificial Intelligence. CIMAF 1999. Special Session on Distributions and Evolutionary Optimization, pp. 332–339 (1999)
7. Hansen, P., Mladenovič, N.: Variable neighborhood search: principles and applications. European Journal of Operational Research 130, 449–467 (2001)
8. Harik, G.: Linkage learning via probabilistic modeling in the ECGA. University of Illinois Genetic Algorithms Laboratory, Urbana (January 1999) IlliGAL Report No. 99010
9. Harik, G.R., Lobo, F.G., Goldberg, D.E.: The compact genetic algorithm. In: Proceedings of the IEEE Conference on Evolutionary Computation 1998 (ICEG 1998), pp. 523–528. IEEE Service Centre, Piscataway (1998)
10. Jarboui, B., Eddaly, M., Siarry, P.: An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems. Computers and Operations Research 36, 2638–2646 (2009)
11. Larrañaga, P., Etxeberria, R., Lozano, J.A., Peña, J.M.: Combinatorial optimization by learning and simulation of Bayesian networks. In: Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence, Stanford, pp. 343–352 (2000)
12. Larrañaga, P., Lozano, J.A. (eds.): Estimation of distribution algorithms: a new tool for evolutionary computation. Kluwer Academic Publishers, Boston (2002)
13. Leisten, R.: Flowshop sequencing problems with limited buffer storage. International Journal of Production Research 28, 2085–3100 (1990)
14. Liu, B., Wang, L., Jin, Y.H.: An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers. Computers and Operations Research 35, 2791–2806 (2008)
15. Lozano, J., Larraanaga, P., Inza, I., Bengoetxea, E.: Towards a New Evolutionary Computation: Advances in the Estimation of Distribution Algorithms. Springer, Heidelberg (2006)
16. Montgomery, D.C.: Design and analysis of experiments, 5th edn. Wiley, New York (2000)
17. Mühlenbein, H., Mahnig, T.: Convergence theory and applications of the factorized distribution algorithm. Journal of Computing and Information Technology 7, 19–32 (1999a)
18. Mühlenbein, H., Mahnig, T.: FDA - A scalable evolutionary algorithm for the optimization of additively decomposed functions. Evolutionary Computation 7, 353–376 (1999b)
19. Mühlenbein, H., Paaß, G.: From Recombination of Genes to the Estimation of Distributions I. Binary Parameters. In: Ebeling, W., Rechenberg, I., Voigt, H.-M., Schwefel, H.-P. (eds.) PPSN 1996. LNCS, vol. 1141, pp. 178–187. Springer, Heidelberg (1996)
20. Nowicki, E.: The permutation flow shop with buffers: a tabu search approach. European Journal of Operational Research 116, 205–219 (1999)
21. Papadimitriou, C.H., Kanellakis, P.C.: Flow shop scheduling with limited temporary storage. Journal of Association Computing Machine 27, 533–549 (1980)
22. Pelikan, M.: Hierarchical Bayesian Optimization Algorithm: Toward a New Generation of Evolutionary Algorithms. Springer, Heidelberg (2005)
23. Pelikan, M., Goldberg, D.E., Cantú-Paz, E.: BOA: The Bayesian optimization algorithm. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 1999, vol. 1, pp. 525–532. Morgan Kaufmann Publishers, San Francisco (1999)
24. Pelikan, M., Mühlenbein, H.: The bivariate marginal distribution algorithm. In: Roy, R., Furuhashi, T., Chawdhry, P.K. (eds.) Advances in Soft Computing Engineering Design and Manufacturing, pp. 521–535. Springer, London (1999)

25. Qian, B., Wang, L., Huang, D.X., Wang, W.L., Wang, X.: An effective hybrid DE-based algorithm for multi-objective flow shop scheduling with limited buffers. Computers and Operations Research 36, 209–233 (2009)
26. Reddi, S.S.: Sequencing with finite intermediate storage. Management Science 23, 216–227 (1976)
27. Reeves, C.R.: A genetic algorithm for flowshop sequencing. Computers and Operations Research 22, 5–13 (1995)
28. Ronconi, D.P.: A branch-and-bound algorithm to minimize the makespan in a flowshop with blocking. Annals of Operations Research 138, 53–65 (2005)
29. Ruiz, R., Maroto, C., Alcaraz, J.: Two new robust genetic algorithms for the flowshop scheduling problem. Omega 34, 461–476 (2006)
30. Tasgetiren, M.F., Sevkli, M., Liang, Y.C., Gencyilmaz, G.: Particle swarm optimization algorithm for permutation flowshop sequencing problem. LNCS, pp. 382–389. Springer, Heidelberg (2004)
31. Wang, L., Zhang, L., Zheng, D.Z.: An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. Computers and Operations Research 33, 2960–2971 (2006)