Computers & Operations Research 39 (2012) 1450–1457

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

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

A Self-guided Genetic Algorithm for permutation flowshop scheduling problems

Shih-Hsin Chen [a], Pei-Chann Chang [b,*], T.C.E. Cheng [c], Qingfu Zhang [d]

[a] Department of Electronic Commerce Management, Nanhua University, Taiwan, ROC
[b] Department of Information Management, Yuan-Ze University, Taiwan, ROC
[c] Department of Logistics and Maritime Studies, The Hong Kong Polytechnic University, 11 Yuk Choi Road, Hung Hom, Kowloon, Hong Kong
[d] School of Computer Science and Electronic Engineering, University of Essex, UK

ARTICLE INFO

Available online 26 August 2011

Keywords:
Estimation of Distribution Algorithms
Permutation flowshop scheduling
Self-guided crossover
Self-guided mutation

ABSTRACT

In this paper we develop a Self-guided Genetic Algorithm (Self-guided GA), which belongs to the category of Estimation of Distribution Algorithms (EDAs). Most EDAs explicitly use the probabilistic model to sample new solutions without using traditional genetic operators. EDAs make good use of the global statistical information collected from previous searches but they do not efficiently use the location information about individual solutions. It is recently realized that global statistical information and location information should complement each other during the evolution process. In view of this, we design the Self-guided GA based on a novel strategy to combine these two kinds of information. The Self-guided GA does not sample new solutions from the probabilistic model. Instead, it estimates the quality of a candidate offspring based on the probabilistic model used in its crossover and mutation operations. In such a way, the mutation and crossover operations are able to generate fitter solutions, thus improving the performance of the algorithm. We tested the proposed algorithm by applying it to deal with the NP-complete flowshop scheduling problem to minimize the makespan. The experimental results show that the Self-guided GA is very promising. We also demonstrate that the Self-guided GA can be easily extended to treat other intractable combinatorial problems.

1. Introduction

Estimation of Distribution Algorithms (EDAs) are a major evolutionary computing paradigm [1–4]. EDAs explicitly build a probabilistic model of promising solutions based on the information extracted from previous searches. They generate new solutions by sampling from the probabilistic model thus built. EDAs provide a systematic way of using global statistical information to guide the search. Solutions sampled by EDAs are more likely to be in a promising area characterized by the probabilistic model. However, EDAs alone do not make good use of the information about the location of individual solutions because the location information is not directly used to guide the search for the optimal solution [5]. Recently some attempts have been made to combine EDAs with the traditional crossover and mutation operators [5–7] with a view to complementing location information and global statistical information during the evolution process. The idea of using the location information about individual solutions to enhance EDAs is supported by the proximate optimality principle, the underlying assumption in modern heuristics.

This principle assumes that good solutions have similar structures or substructures [8]. This assumption is valid in many real-world applications [9]. For example, the percentage of common edges in any two locally optimal solutions for a traveling salesman problem obtained by the Lin–Kernighan method is about 85% on average [10]. Experimental results reported in [6] also confirmed that this assumption holds for a single-machine scheduling problem. This principle suggests that the location information about good solutions should not be ignored in the design of an efficient search method.

In this paper we propose a Self-guided Genetic Algorithm (Self-guided GA), which is based on a novel method to combine global statistical information with the location information about individual solutions to deal with intractable combinatorial optimization problems. In our proposed algorithm, we use global statistical information to guide the crossover and mutation operators, instead of directly using it to sample new solutions. More specifically, we employ an approximate probabilistic model to estimate the quality of candidate solutions and to enable the crossover and mutation operators to generate more promising

* Corresponding author. Tel.: +886 34638909; fax: +886 34638884.
E-mail addresses: shihhsin@mail.nhu.edu.tw (S.-H. Chen), iepchang@saturn.yzu.edu.tw (P.-C. Chang), LGTcheng@polyu.edu.hk (T.C.E. Cheng), qzhang@essex.ac.uk (Q. Zhang).

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2011.08.016

solutions. Although the probabilistic model has been utilized to predict the chromosome fitness [8,11–14] or Kacem et al. [15] proposed the fuzzy logic to guide the weights in the multi-objective problems, they were not used in the same way to guide the genetic operators.

NP-complete combinatorial optimization problems occur in many real-world applications. It is generally believed that there is no polynomial-time algorithm for finding the best solutions to these problems. For this reason, heuristics including evolutionary algorithms are widely used to find reasonably good solutions to NP-complete problems. Permutation flowshop scheduling problem (PFSP) was studied in this paper and PFSP is one of the best known NP-complete problems. Excellent reviews on this problem can be found in [16,17]. Several evolutionary algorithms have been proposed to deal with this problem [18–22]. In this paper we apply the Self-guided GA to treat this problem as a way to illustrate how the algorithm works.

The rest of the paper is organized as follows: Section 2 introduces the permutation flowshop scheduling problem and EDAs. Section 3 provides detailed explanation of the Self-guided GA. Section 4 presents experimental results on the performance of the proposed algorithm in treating the permutation flowshop scheduling problem to minimize the makespan. Section 5 concludes the paper.

2. Literature review and problem definition

Section 2.1 presents the definition of the permutation flowshop problem. Section 2.2 briefly reviews the literature on flowshop scheduling. While the Self-guided GA belongs to the category of EDAs, it is very different from other EDA approaches. In order to highlight the differences between our work and previous EDA research, we review EDAs in Section 2.3 and point out the differences between our proposed algorithm and other EDAs.

2.1. Problem definition of flowshop scheduling

Flowshops provide a convenient means to model serial manufacturing processes. The flowshop is a processing facility that consists of several machines on which jobs are processed in a sequential manner. In the permutation flowshop problem (PFSP), all the jobs follow the same processing order on each of the machines and every jobs are independent to each other.

The PFSP to minimize the makespan can be defined as follows.

Suppose there are n jobs and m machines. Let $p(i,j)$, $1 \le i \le n$, $1 \le j \le m$be the processing time of job i on machine j and $\pi = (\pi_1, \ldots, \pi_n)$ be a job permutation (i.e., processing order of the jobs). Then the completion times $C(\pi_i,j)$ are calculated as follows:

$$C(\pi_1,1) = p(\pi_1,1), \tag{1}$$

$$C(\pi_i,1) = C(\pi_{i-1},1) + p(\pi_i,1) \quad \text{for } i = 2,\ldots,n, \tag{2}$$

$$C(\pi_1,j) = C(\pi_1,j-1) + p(\pi_1,j) \quad \text{for } j = 2,\ldots,m, \tag{3}$$

$$C(\pi_i,j) = \max\{C(\pi_{i-1},j), C(\pi_i,j-1)\} + p(\pi_i,j) \quad \text{for } i = 2,\ldots,n;\; j = 2,\ldots,m. \tag{4}$$

The makespan is

$$C_{\max}(\pi) = C(\pi_n,m). \tag{5}$$

The objective is to find a permutation π^* that minimizes $C_{\max}(\pi)$.

2.2. Literature review of flowshop scheduling

Flowshop scheduling is one of the most extensively studied problems in the area of scheduling. Ref. [23] summarized the assumptions for the PFSP and presented results on some solvable cases. Since most variants of the PFSP are NP-complete, research on the PFSP has been focused on developing effective heuristics.

Ref. [16] provided a review and a classification of the heuristics for the PFSP. Ref. [17] presented a comprehensive survey of the major results on this problem over the period from 1954 to 2004. The survey covered exact methods and heuristic methods (including evolutionary algorithms). They also provided a good reference for the PFSP with the objective of minimizing the makespan, denoted as $n/m/p/C_{\max}$. Ref. [24] provided a comprehensive review and evaluation of the heuristics for the PFSP. It is clear that heuristics have been a major methodology to deal with the PFSP. The reader may refer to [24] for a detailed review of metaheuristics, including tabu search, simulated annealing, genetic algorithms, iterated local search, and hybrid techniques, for tackling various flowshop scheduling problems.

As regards research on using EDAs, [20] employed ACGA (artificial chromosomes embedded in genetic algorithm), which is a hybrid framework of EDAs and GAs, to tackle the PFSP to minimize the makespan. Their results show that ACGA performs better than GAs and GADP (genetic algorithm with dominance properties) [25]. In [21], they incorporated the order information about each job into the probabilistic model to solve the PFSP to minimize the total flowtime. They also used a variable neighborhood search in the algorithm as an improvement procedure. Their approach outperformed all the other algorithms in their comparison study.

2.3. Estimation of Distribution Algorithms

EDAs also called evolutionary algorithms with probabilistic models (i.e., EAPMs) or probabilistic model-building genetic algorithms (PMBGAs) [4] is a principled alternative to traditional evolutionary algorithms. EDAs do not use crossover or mutation. Instead, they explicitly extract global statistical information from the previous search and build a posterior probabilistic model of promising solutions, from which new solutions are sampled.

Population-based incremental learning (PBIL) [26,27], Univariate Marginal Distribution Algorithm (UMDA) [3], and compact genetic algorithm (cGA) [2] may be the simplest versions of EDAs. These algorithms do not consider variable interactions in modeling. Theoretical studies have shown that these EDAs are unable to solve some hard problems with complicated variable linkages [28]. Much effort has been made to improve these simple EDAs. The improvements could be classified into three categories:

- Use of high order variable interactions in probabilistic models. Examples are combined optimizers with the mutual information tree (COMIT) [29], dependency-trees [30], and the Bayesian optimization algorithm (BOA) [31]. Although some promising results have been reported, [5] showed that models with high order variable interactions do not necessarily outperform simple models in dealing with some real-world hard problems because these complicated models can only consider a very tiny percentage of variable interactions in a hard problem.
- Combination of EDAs with traditional evolutionary algorithms. Such combinations aim at taking advantage of both EDAs and other EAs. For example, in EDA-GA in [7], some solutions are sampled from the probabilistic model and the others are produced by crossover and mutation. In [6], artificial chromosomes are generated by an EDA and then mixed with generic

chromosomes. In guided mutation [5], part of a new solution is copied from its parent and the rest is sampled from a probabilistic model.

- Hybridization with local search. Encouraged by the success of combining traditional EAs and local search, many attempts have been made to enhance the performance of EDAs by local search. For example, cGA with the Lin–Kernighan local search has been used to deal with large-scale traveling salesman problems [1]. EA/G with a simple local search heuristics [5] has been proposed for the maximum clique problem. An EDA with local search in [32] has been designed for the nurse rostering problem.

Our proposed algorithm falls into the second category. The proposed approach works better than other existing ones. That is because the proposed approach will evaluate the fitness predictor before applying cross-over or mutation operator. Thus, redundant or unnecessary moves can be prevented.The difference between our algorithm and previous work in this category is that the probabilistic model is not a source for generating new solutions, but acts as a fitness predictor for guiding the crossover and mutation operators to generate fitter solutions. Although the probabilistic model has been employed to serve as a fitness predictor before (e.g. [8,11–14]), no prior approaches used the probabilistic model in the same way as that used in our proposed algorithm.

3. Algorithm

The Self-guided GA uses the probabilistic model to guide its crossover and mutation operators. The probabilistic model is used to estimate the quality of candidate solutions generated by the traditional crossover and mutation operators. Those with estimated higher quality will be allowed to enter the next generation for further evolution. In such a way, both global statistical information and location information about individual solutions (i.e., parent solutions in crossover and mutation) are used to produce new solutions.

The procedure of the Self-guided GA is described as follows:

Notation:

- *Population*: a set of solutions.
- *PopSize*: the size of *Population*.
- G: the maximum number of generations.
- t: generation index.
- $P(t)$: probabilistic matrix at generation t.
- *Parentset*: the set of parent solutions selected from the current population.
- *Newset*: the set of new solutions generated at each generation.
- N: the size of *Newset*.

Algorithm.

```
1:  Initialize Population
2:  Evaluate(Population)
3:  $t \leftarrow 0$
4:  Initialize $P(t)$
5:  while $t < G$ do
6:      Parentset=Select(Population)
7:      $P(t+1) \leftarrow$ modelupdate(Parentset, $P(t)$)
8:      Generate Newset by using self-guided crossover
9:      Mutate every new solution in Newset by using self-
        guided mutation
10:     Evaluate(Newset)
11:     Use the solutions in Newset to replace the $N$ worst
        solutions in Population
12:     $t \leftarrow t+1$
13: end while
```

In Line 1 of the algorithm, a set of solutions is generated randomly or by using a heuristic to form the initial *Population*, and the objective function values (i.e., their makespan values) of all the solutions in *Population* are computed in Line 2. Line 3 initializes the generation index t. The probability matrix $P(t)$ is initialized in Line 4. Line 6 selects a number of solutions from *Population* to form *Parentset*. Detailed selection method can be referred to Section 3.1. Line 7 computes $P(t+1)$. In Line 8, we employ the self-guided crossover operator (the details of this crossover are given in Section 3.5) to produce N solutions. In Line 9, we mutate each new solution generated in Line 8 by using the self-guided mutation operator (the details are given later). Line 10 computes the objective function values of all the new solutions. In Line 11, new solutions replace the N worst solutions in *Population* and the *Popsize*$-N$ best solutions in the old population will enter the new population. This elitism scheme is widely used in evolutionary algorithms controlled by an elitism ratio.

In the following we give the details of our proposed algorithm.

3.1. Selection

In principle, any selection method such as proportional selection and tournament one can be used for this purpose (Line 6). For the sake of simplicity, we adopted the 2-tournament selection in our experiments. It randomly draws two members from *Population* and selects the fitter one to become a member in *Parentset*. As a result, a number of independent 2-tournament selections are conducted to generate *Parentset*.

3.2. Probabilistic model

Probability $P(t)$ is of the form:

$$P(t) = \begin{pmatrix} P_{11}(t) & \cdots & P_{1n}(t) \\ \vdots & \ddots & \vdots \\ P_{n1}(t) & \ldots & P_{nn}(t) \end{pmatrix}, \tag{6}$$

where $P_{ij}(t)$ is the probability of job i in position j in a promising solution. $P(t)$ summarizes the global statistical information about promising solutions obtained from the previous search.

In Line 4, each $P_{ij}(t)$ is initialized to be $1/n$, where n is the total number of jobs in $|Parentset|$. This initialization means that all the solutions have the same likelihood to be an optimal solution. The reason for such an initialization is that we have no information about the location of promising solutions.

Let ϕ_{ij} be the number of the solutions in *Parentset* in which job i is in position j and $|Parentset|$ be the size of *Parentset*. $P_{ij}(t+1)$ in Line 7 is updated as follows:

$$P_{ij}(t+1) = (1-\lambda)P_{ij}(t) + \lambda \frac{\phi_{ij}+1}{|Parentset|+n}, \tag{7}$$

$\phi_{ij}/|Parentset|$ is the percentage of solutions in which job i is in position j. It represents knowledge of promising solutions learnt from the current generation. We use $(\phi_{ij}+1)/(|Parentset|+n)$, the Laplace correction of $\phi_{ij}/|Parentset|$ in Eq. (7), to prevent P_{ij} from becoming very small [33–35]. $P_{ij}(t)$ is historical knowledge of promising solutions. We update $P(t+1)$ in an incremental manner which is suggested by [29]. $\lambda \in (0,1)$ balances the contribution from historical knowledge with that from the knowledge learnt from the current generation.

3.3. Estimation of solution quality

With $P(t+1)$, we define the following function to estimate the quality of a solution X:

$$Q_{t+1}(X) = \prod_{k=1}^{n} P_{k[k]}(t+1), \quad (8)$$

where $[k]$ is the position of job k in X. We would like to make the following remarks on this function:

- $P_{k[k]}(t+1)$ is the probability that job k in position $[k]$ is a promising solution. Therefore $Q_{t+1}(X)$ can measure how 'promising' X is.
- In general, $Q_{t+1}(X)$ is not an exact probability measure on the set of all the solutions X since
  $$\sum_X Q_{t+1}(X) \neq 1.$$
  $Q_{t+1}(X)$ is just an estimation value of the probability that X is promising. This estimation is much easier to compute and more effective when compared with other probabilistic models in the literature [8,11–14]. The trade-off among our proposed approach and theirs is the computational efforts needed in evaluating the new chromosome generated. Our approach is simple and effective and save lots of computational times.

We use $Q_{t+1}(X)$ to guide crossover and mutation. In the following we drop $t+1$ in P and Q for simplicity.

3.4. Self-guided mutation operator

Let X be a solution (i.e., a permutation from 1 to n) to be mutated. Let the position of job k be $[k]$ in X. Suppose we swap the positions of jobs i and j in X and obtain Y. We can compute the quality difference in Q caused by this swap as follows:

$$\Delta_{ij} = Q(Y) - Q(X) = [(P_{i[j]}P_{j[i]}) - (P_{i[i]}P_{j[j]})] \prod_{k \neq i,j} P_{k[k]}. \quad (9)$$

The larger Δ_{ij} is, the more likely that Y is better than X.

In our proposed self-guided mutation operator, we randomly pick some pairs of jobs. This parameter is named TM which sets the number of swap operations in our proposed algorithm. For each pair, we compute the differences in Q caused by swapping the positions of its two jobs in the parent solution X. Then we conduct the swapping with the largest difference in X to produce a new solution. Formally, the self-guided mutation operator works as follows:

Notation:

- X: parent solution, i.e., the solution to be mutated.
- TM: the number of swappings to be considered.
- Y: offspring, i.e., the new solution.

Self-guided mutation:

1. Randomly select TM pairs of jobs: $\{i_1,j_1\},\ldots,\{i_{TM},j_{TM}\}$.
2. Compare $\Delta_{i_1j_1},\ldots,\Delta_{i_{TM}j_{TM}}$ and find the pair $\{i_k,j_k\}$ with the largest Δ value.
3. Swap the positions of jobs i_k,j_k in X to generate Y.

In the self-guided mutation operator, we need to compare TM quality differences. In the following we give a simple way to compare two differences.

- **Case 1:** when i,j,l, and m are four different jobs. Given
  $$\Delta_{ij} - \Delta_{lm} = (P_{i[j]}P_{j[i]}P_{l[l]}P_{m[m]} - P_{i[i]}P_{j[j]}P_{l[m]}P_{m[l]}) \times \prod_{k \neq i,j,l,m} P_{k[k]},$$
  $$\Delta_{ij} > \Delta_{lm},$$
  if and only if
  $$P_{i[j]}P_{j[i]}P_{l[l]}P_{m[m]} - P_{i[i]}P_{j[j]}P_{l[m]}P_{m[l]} > 0. \quad (10)$$
- **Case 2:** when i,j, and l are three different jobs. Given
  $$\Delta_{ij} - \Delta_{j,l} = (P_{i[j]}P_{j[i]}P_{l[l]} - P_{i[i]}P_{j[l]}P_{l[j]}) \times \prod_{k \neq i,j,l} P_{k[k]},$$
  $$\Delta_{ij} > \Delta_{jl,}$$
  if and only if
  $$P_{i[j]}P_{j[i]}P_{l[l]} - P_{i[i]}P_{j[l]}P_{l[j]} > 0. \quad (11)$$

Based on the above results, we can use (10) and (11) to compare Q differences in the self-guided mutation operator according to the largest Δ value.

3.5. Self-guided crossover operator

In this paper we propose an approach to using the quality function Q to guide the two-point center crossover [36]. Our approach can be easily extended to other crossover operators.

The two-point center crossover for permutation vectors from 1 to n works as follows:

Notation:

- $X=(x_1,\ldots,x_n)$: parent solution 1.
- $Y=(y_1,\ldots,y_n)$: parent solution 2.
- $Z=(z_1,\ldots,z_n)$: offspring, i.e., the new solution generated by crossover.

Two-point crossover:

1. Randomly select two crossover points $K<L$.
2. Set
   $$z_i = x_i \quad (12)$$
   for $i=1,\ldots,K-1,L+1,\ldots,n$.
3. Reorder $x_K,\ldots,x_L$ in the order that they appear in permutation Y: $u_K,u_{K+1},\ldots,u_L$.
4. Set
   $$z_i = u_i \quad (13)$$
   for $i=K,\ldots,L$.

Now we give an example to illustrate how this crossover works. Suppose that

$$X=(1\ 3\ 2\ 6\ 5\ 4\ 7\ 9\ 8),$$

$$Y=(6\ 4\ 5\ 1\ 2\ 3\ 8\ 9\ 7),$$

and the two crossover points are 3 and 5. Then

$$z_1=x_1=1, \quad z_2=x_2=3, \quad z_6=x_6=4,$$

$z_7 = x_7 = 7, \quad z_8 = x_8 = 9, \quad z_9 = x_9 = 8.$

The order of $x_3 = 2, x_4 = 6, x_5 = 5$ in Y is

6, 5, 2.

Thus

$z_3 = 6, \quad z_4 = 5, \quad z_6 = 2.$

Therefore, the offspring Z is

$Z = (1\ 3\ 6\ 5\ 2\ 4\ 7\ 9\ 8).$

In the two-point crossover, the Q difference between offspring Z and parent 1 X is

$$\Phi(Z,X) = Q(Z) - Q(X) = \left[\prod_{K \le k \le L} P_{y_i i} - \prod_{K \le k \le L} P_{x_i i}\right] \times \prod_{1 \le i < K} P_{x_i i} \prod_{L < i \le n} P_{x_i i}. \tag{14}$$

The larger $\Phi(Z,X)$ is, the more likely that Z is better than X.

The self-guided two-point crossover works as follows:

Notation:

- X: parent solution 1.
- TC: the number of candidates for parent 2.
- Y^j: $j = 1, \ldots, TC$: candidates for parent 2.
- Z: offspring, i.e., the new solution generated by crossover.

Self-guided two-point crossover:

```
1. Randomly select two crossover points $K < L$.
2. for $j = 1$ to $TC$ do
3.    Let $X$ be parent 1 and $Y^j$ be parent 2, and the crossover points be $K$ and $L$. Perform two-point crossover on $X$ and $Y^j$ and generate $Z^j = (z_1^j, \ldots, z_n^j)$.
4. end for
5. Compare $\Phi(Z^i,X)$, $(i = 1, \ldots, TC)$. Output the $Z^i$ with the largest $\Phi$ value as $Z$.
```

In this self-guided two-point crossover, we need to compare $\Phi(Z^i,X)$.

Given

$$\Phi(Z^l,X) - \Phi(Z^m,X) = \left[\prod_{K \le k \le L} P_{z_i^l i} - \prod_{K \le k \le L} P_{z_i^m i}\right] \times \prod_{1 \le i < K} P_{x_i i} \prod_{L < i \le n} P_{x_i i}, \tag{15}$$

$\Phi(Z^l,X) > \Phi(Z^m,X)$

if and only if

$$\prod_{K \le k \le L} P_{z_i^l i} > \prod_{K \le k \le L} P_{z_i^m i}. \tag{16}$$

We can use (16) in the self-guided two-point crossover to compare $\Phi(Z^i,X)$ which is to select a better candidate for parent 2. The time complexity of finding the largest Φ is $O(n \times TC)$.

The Self-guided GA is different from previous EDAs, which explicitly sample new solutions without using the crossover and mutation operators. The Self-guided GA embeds the probabilistic model in the crossover and mutation operators to explore and exploit the solution space. The following section demonstrates the performance of the Self-guided GA in the flowshop scheduling problems.

4. Experimental studies

We conducted extensive computational experiments to compare the Self-guided GA with several other algorithms for the PFSPs by using the 110 Taillard instances [37]. The test problem was the PFSP to minimize the makespan as defined in Section 2. It involves processing n jobs on m machines. There were 10 instances in each of two different combinations of (n,m). In combination one, $n = 20, 50, 100$ and $m = 5, 10, 20$ whereas in combination two, $(n = 200, m = 10)$ and $(n = 200, m = 20)$. We coded all the algorithms in Java 2 and performed the experiments on a Windows 2003 server (Intel Xeon 3.2 GHZ). The following sections described the brief concept of the compared algorithms and their own parameter settings.

4.1. Algorithms for comparison and parameter settings

In our experiments, we compared seven GA-based algorithms, namely Self-guided GA, SGA, ACGA, GMA, and MGGA. We ran each algorithm 30 times independently on each test instance and there were a total of 110 instances. We obtained the experimental results based on these tests. The brief explanation of the five compared algorithms was shown as follows:

- SGA: A standard genetic algorithm. It is the same as the Self-guided GA except that it is not augmented with the probabilistic model. Comparison with SGA helps us to understand the effect of the probabilistic model in the Self-guided GA. We coded the algorithm ourselves.
- ACGA [38]: Artificial Chromosome with Genetic Algorithms. This is a recently developed approach that combines an EDA with a traditional algorithm. The probabilistic model and genetic operators are used to generate new solutions in a different way from that used in our proposed Self-guided GA. By using the hybrid framework, both global and local information are used. This algorithm was developed in the laboratory of the second author. We used the same code as in [38].
- GMA [11]: Guided Memetic Algorithm. The probabilistic model is used to assist a local search operator to reduce the computational overhead. This algorithm was developed in the laboratory of the second author. We used the same code as in [11].
- MGGA [38]: Mining Gene Genetic Algorithms. This algorithm was designed for treating machine scheduling problems. The linear assignment algorithm and a greedy heuristic are embedded in MCGA. This algorithm was developed in the laboratory led by the second author of this paper. We used the same code as in [38].
- PSO_{spv} [39]: It is a particle swarm optimization designed for treating the PFSP. We used the data from [39] for comparison.
- CPSO: This algorithm was designed by [40], which employs a heuristic as an improvement procedure. We used the data from [39] for comparison.
- DDE [41]: The Discrete Differential Evolution. It is a method designed for treating the PFSP. A problem-specific heuristic NEH [42] is used in DDE. We used the data from [41] for comparison.

Except the CPSO and DDE, there are five algorithms conducted by the authors of this paper. In order to do a fair comparison, the parameter settings were tuned systematically by Design-of-Experiment (DOE). DOE is a statistic method which provides a systematic way to distinguish the effects of the factors and the difference among or between different levels [43]. There are also different methods that can be applied to learn these parameters. Interested readers refer to [44]. The parameter settings of the five algorithms are given in Table 1.

To be fair to all the algorithms, we terminated each algorithm after $500 \times 2 \times n$ evaluations of the objective function. We should point out that the same stopping criterion was used in PSO_{SPV}, CPSO, and DDE in [39–41], respectively.

4.2. Experimental results

4.2.1. Is the self-guided strategy useful?

To address this issue, we present the experimental results of the average error ratio (*ER*) of the compared algorithms. In the literature, *ER* is often used to evaluate the performance of algorithms applied to deal with the PFSPs, whereby the error ratio of a solution X_i generated by an algorithm is calculated as follows:

$$ER_i = \frac{C_{\max}(X_i) - U_i}{U_i},$$

where U_i is the makespan value of the best known or optimal solution provided by [37]. Table 2 shows the statistics of the average *ER* values of all the algorithms on all the 110 test instances. The results show that the Self-guided GA outperformed SGA, PSO_{spv}, CPSO, and GMA in terms of the *ER* value. It is also evident that the Self-guided GA performed worse than DDE. It is because that DDE employs the problem-specific heuristics NEH in permutation flowshop scheduling problems while the Self-guided GA does not use any domain knowledge. This reason implies that the performance of the Self-guided GA could be further improved if the NEH heuristic is utilized. We will study the effect of the NEH heuristic on the Self-guided GA in the future.

Finally, even although the performance of Self-guided GA, ACGA, and MGGA is quite similar, it is not necessary that the three algorithms performed equally because the average value is often influenced by the extreme values [43]. For this we used the Analysis of Variance (ANOVA) to distinguish the real performance in the next subsection.

4.2.2. Statistical tests

To perform a statistically sound analysis of the experimental results, we conducted ANOVA on the objective function values of the final solutions obtained for all the test instances by all the five GA-based algorithms that we ran in our experiments. We are particularly interested in the performance of Self-guided GA, ACGA, and MGGA because their average error ratio is similar. The ANOVA results are presented in Table 3. In the ANOVA table, the source indicates factors and combinations of factors. In our case, *Instance* and *Method* are factors. *DF* represents the degree of freedom and *SS* is the sum of squares. The mean square is equal to *SS* divided by *DF*. If the *Pr*-value of a factor (source) is less than

Table 1
Parameter settings of the implemented algorithms: Self-guided GA, SGA, ACGA, GMA, and MGGA.

Method	Settings
Self-guided GA	TC=4 TM=2 $\lambda=0.5$ Population size=100 Parentset size=100
SGA	Crossover rate=0.6 Mutation rate=0.3 Population size=100
ACGA	Starting generation=0.7*(total generations) Interval=0.1*(total generations) Crossover rate=0.6 Mutation rate=0.3 $\lambda=0.5$ Population size=500
GMA	Crossover rate=0.6 Mutation rate=0.3 $\lambda=0.5$ Population size=100
MGGA	Interval: 0.05*(total generations) Crossover rate=0.9 Mutation rate=0.5 Population size=100
Common settings	The top 10% solutions in current population directly enter the next generation

Table 2
Average error ratios of all the algorithms on Taillard's instances.

n	*m*	SGA	MGGA	ACGA	Self-guided GA	PSO_{spv}	*DDE*	CPSO	GMA
20	5	1.02	0.81	1.08	1.10	1.75	0.46	1.05	1.14
	10	1.73	1.4	1.62	1.90	3.25	0.93	2.42	2.30
	20	1.48	1.06	1.34	1.60	2.82	0.79	1.99	2.01
50	5	0.61	0.44	0.57	0.52	1.14	0.17	0.90	0.47
	10	2.81	2.56	2.79	2.74	5.29	2.26	4.85	3.21
	20	3.98	3.82	3.75	3.94	7.21	3.11	6.40	4.97
100	5	0.47	0.41	0.44	0.38	0.63	0.08	0.74	0.42
	10	1.67	1.5	1.71	1.60	3.27	0.94	2.94	1.96
	20	3.8	3.15	3.47	3.51	8.25	3.24	7.11	4.68
200	10	0.94	0.92	0.94	0.80	2.47	0.55	2.17	1.10
	20	2.73	3.95	2.61	2.32	8.05	2.61	6.89	3.61
	Total	1.93	1.82	1.85	1.85	4.01	1.38	3.40	2.35

Table 3
ANOVA results on the objective function values of the final solutions obtained by the five algorithms on the 110 Instances.

Source	DF	SS	Mean square	*F* value	Pr > *F*
Instances	109	1.924×10^{11}	1,765,744,182	1,976,058.93	<0.0001
Algorithm	4	2,137,240	534,310	597.95	<0.0001
Instances*algorithm	436	10,234,102	23,473	26.27	<0.0001
Error	15,950	14,252,419	894		
Corrected total	16,499	1.925×10^{11}			

Table 4
Duncan post-hoc test on the five different algorithms (the objective function values of the final solutions).

Duncan grouping	Mean	*N*	Method
A	5035.06	3300	GMA
B	5016.3655	3300	MGGA
C	5011.263	3300	SGA
D	5007.45	3300	ACGA
E	5003.5873	3300	Self-guided GA

Table 5
ANOVA results on the CPU time used by the five algorithms for the 110 instances.

Source	DF	SS	Mean square	*F* value	$\Pr > F$
Instances	109	3,485,410	31,976	47,283.04	<0.0001
Method	4	685,179	171,295	253,292.37	<0.0001
Instances∗method	436	1,632,899	3745	5537.97	<0.0001
Error	15,950	10,787	1		
Corrected total	16,499	5,814,275			

Table 6
Duncan post-hoc test of the five different algorithms (CPU time).

Duncan grouping	Mean	*N*	Method
A	22.70	3300	MGGA
B	9.18	3300	Self-guided GA
C	6.80	3300	ACGA
D	6.53	3300	SGA
E	5.39	3300	GMA

0.05, it means that there is a significant difference in this factor [43].

In Table 3, the factor *method* has a *Pr*-value less than 0.0001. Therefore, all these five GA-based algorithms performed very differently. We further conducted the Duncan grouping test to differentiate the performance of these algorithms. The Duncan test results were shown in Table 4. In this table, mean is the average value and *N* is the number of the observations. In the Duncan grouping test, if two algorithms share the same alphabet (i.e., they are in the same group), there is no significant difference between them. Otherwise they are significantly different [43]. It is evident from Table 4 that the five algorithms were significantly different in their performance and the Self-guided GA performed significantly better than the others.

We further conducted ANOVA and the Duncan grouping tests on the CPU times of the five algorithms. The results are given in Tables 5 and 6. Clearly the Self-guided GA was much faster than MGGA but a bit slower than the other three algorithms.

In conclusion, in terms of solution quality, the Self-guided GA is significantly better than the other four GA-based algorithms. However, the Self-guided GA may consume more CPU time.

5. Discussions and conclusions

When we compared the Self-guided GA with SGA, SGA is the same as the Self-guided GA except that SGA does not use global statistical information to guide the search. It is clear from Table 4 that the Self-guided GA outperformed SGA in terms of the Duncan grouping test. These results suggest that global statistical information does improve the algorithm performance significantly. On the other hand, ACGA uses both global statistical information and location information about individual solutions. However, to let the probabilistic models guide the search might be better. It is because Table 4 also shows that the Self-guided GA was much better than ACGA. Thus, we may conclude that the strategy of combining two kinds of information in the Self-guided GA is more effective and efficient than SGA and ACGA.

To conclude the research, we introduced and explored a new strategy for combining global statistical information about promising search areas and location information about individual solutions. The strategy uses global statistical information to estimate the quality of a candidate solution for guiding the crossover and mutation operations. We employed this new strategy in this research to develop the Self-guided GA. We assessed the performance of the proposed algorithm in dealing with the NP-complete permutation flowshop scheduling problem to minimize the makespan. The experimental results show that the Self-guided GA indeed performs well in comparison with several other GA-based algorithms published in the literature. Our work can readily be extended to deal with other intractable combinatorial optimization problems.

Future topics along this line of research may include:

- Enhancing the proposed algorithm by using problem-specific knowledge. For example, NEH is suitable for the PFSP.
- Initializing the probabilistic model by an approach instead of using $1/n$ for each $P_{ij}(t)$. This method could be based on the values of the makespan and on the number of times we have job *j* scheduled at position *i*.
- Extending the idea to combining EDAs with other meta-heuristics and local search such as tabu search and variable neighborhood search.
- Broadening the approach to deal with multi-objective optimization problems.
- Taking the variable interactions in the probabilistic models.

References

[1] Baraglia R, Hidalgo J, Perego R. A hybrid heuristic for the traveling salesman problem. IEEE Transactions on Evolutionary Computation 2001;5(6):613–22.

[2] Harik G, Lobo F, Goldberg D. The compact genetic algorithm. IEEE Transactions on Evolutionary Computation 1999;3(4):287–97.

[3] Muhlenbein H, Paaß G. From recombination of genes to the estimation of distributions I. Binary parameters. In: Lecture Notes in Computer Science, vol. 1141; 1996. p. 178–87.

[4] Pelikan M, Goldberg DE, Lobo FG. A survey of optimization by building and using probabilistic models. Computational Optimization and Applications 2002;21(1):5–20.

[5] Zhang Q, Sun J, Tsang E. An evolutionary algorithm with guided mutation for the maximum clique problem. IEEE Transactions on Evolutionary Computation 2005;9(2):192–200.

[6] Chang PC, Chen SH, Fan CY. Mining gene structures to inject artificial chromosomes for genetic algorithm in single machine scheduling problems. Applied Soft Computing Journal 2008;8(1):767–77.

[7] Pena J, Robles V, Larranaga P, Herves V, Rosales F, Perez M. Ga-eda: hybrid evolutionary algorithm using genetic and estimation of distribution algorithms. In: Orchard B, Yang C, Ali M, editors. Innovations in applied artificial intelligence, Lecture notes in computer science, vol. 3029. Berlin/Heidelberg: Springer; 2004. p. 361–71.

[8] Lima C, Pelikan M, Lobo F, Goldberg D. Loopy substructural local search for the Bayesian optimization algorithm, engineering stochastic local search algorithms. Designing, Implementing and Analyzing Effective Heuristics 2009:61–75.

[9] Glover F, Kochenberger G. Critical event tabu search for multidimensional knapsack problems. Kluwer Academic Publishers; 1996. p. 407–27.

[10] Lin S, Kernighan B. An effective heuristic algorithm for the traveling-salesman problem. Operations Research 1973;21(2):498–516.

[11] Chen SH, Chang PC, Zhang Q, Wang CB. A guided memetic algorithm with probabilistic models. International Journal of Innovative Computing, Information and Control 2009;5(12):4753–64.

[12] Brownlee AEI, McCall JAW, Zhang Q, Brown DF. Approaches to selection and their effect on fitness modelling in an estimation of distribution algorithm. In: IEEE congress on evolutionary computation, 2008. CEC 2008. IEEE; 2008. p. 2621–8.

[13] Sastry K, Pelikan M, Goldberg D. Efficiency enhancement of genetic algorithms via building-block-wise fitness estimation. In: Congress on evolutionary computation, 2004, vol. 1. IEEE; 2004. p. 720–7.

[14] Sastry K, Pelikan M, Goldberg D. Efficiency enhancement of estimation of distribution algorithms. Computational Intelligence 2006;33:161–85.

[15] Kacem I, Hammadi S, Borne P. Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. Mathematics and Computers in Simulation 2002;60(3–5): 245–76.

[16] Framinan JM, Gupta JND, Leisten R. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. Journal of the Operational Research Society 2004;55(12):1243–55.

[17] Hejazi S, Saghafian S. Flowshop-scheduling problems with makespan criterion: a review. International Journal of Production Research 2005;43(14): 2895–2929.

[18] 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.

[19] Reeves CR. A genetic algorithm for flowshop sequencing. Computers and Operations Research 1995;22(1):5–13.

[20] Chang P, Chen SH, Fan CY. Generating artificial chromosomes by mining gene structures with diversity preservation for scheduling problems. Annals of Operations Research 2010;180(1):197–211.

[21] 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 2009;36(9):2638–46.

[22] Minella G, Ruiz R, Ciavotta M. A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. INFORMS Journal on Computing 2008;20(3):451–71.

[23] Baker KR. Introduction to sequencing and scheduling. Wiley; 1974.

[24] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research 2005;165(2):479–94.

[25] Chang PC, Chen SH, Mani V. A hybrid genetic algorithm with dominance properties for single machine scheduling with dependent penalties. Applied Mathematical Modeling 2009;33(1):579–96.

[26] Baluja S. Population-based incremental learning: a method for integrating genetic search based function optimization and competitive learning. PA, USA: Carnegie Mellon University Pittsburgh; 1994.

[27] Baluja S. An empirical comparison of seven iterative and evolutionary function optimization heuristics. Technical Report CMU-CS-95-193, Carnegie Mellon University; 1995.

[28] Zhang Q, Muhlenbein H. On the convergence of a class of estimation of distribution algorithms. IEEE Transactions on Evolutionary Computation 2004;8(2):127–36.

[29] Baluja S, Davies S. Fast probabilistic modeling for combinatorial optimization. In: Proceedings of the fifteenth national/tenth conference on artificial intelligence. John Wiley & Sons Inc.; 1998. p. 469–76.

[30] Baluja S, Davies S. Using optimal dependency-trees for combinational optimization. In: Proceedings of the fourteenth international conference on machine learning table of contents; 1997. p. 30–8.

[31] Pelikan M, Goldberg DE, Cantu-Paz E. BOA: the Bayesian optimization algorithm. In: Proceedings of the genetic and evolutionary computation conference GECCO-99, vol. 1; 1999. p. 525–32.

[32] Aickelin U, Burke EK, Li J. An estimation of distribution algorithm with intelligent local search for rule-based nurse rostering. Journal of the Operational Research Society 2007;58(12):1574–85.

[33] Good I. The estimation of probabilities: an essay on modern Bayesian methods. The MIT Press; 2003.

[34] Han J, Kamber M. Data mining: concepts and techniques. Morgan Kaufmann; 2006.

[35] Cestnik B. Estimating probabilities: a crucial task in machine learning. In: Proceedings of the ninth european conference on artificial intelligence; 1990. p. 147–9.

[36] Murata T, Ishibuchi H. Performance evaluation of genetic algorithms for flowshop scheduling problems 1994; 812–7.

[37] Taillard E. Benchmarks for basic scheduling instances. European Journal of Operational Research 1993;64:278–85.

[38] Chang PC, Chen SH, Fan CY. Genetic algorithm integrated with artificial chromosomes for multi-objective flowshop scheduling problems. Applied Mathematics and Computation 2008;205(2):550–61.

[39] Tasgetiren M, Liang Y, 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.

[40] Jarboui B, Ibrahim S, Siarry P, Rebai A. A combinatorial particle swarm optimisation for solving permutation flowshop problems. Computers & Industrial Engineering 2008;54(3):526–38.

[41] Pan Q, Tasgetiren M, Liang Y. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Computers & Industrial Engineering 2008;55(4):795–816.

[42] Nawaz M, Enscore E, Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA 1983;11(1):91–5.

[43] Montgomery D. Design and analysis of experiments. John Wiley & Sons Inc.; 2008.

[44] Eiben A, Schut M. New ways to calibrate evolutionary algorithms. In: Siarry P, Michalewicz Z, editors. Advances in metaheuristics for hard optimization, natural computing series. Springer; 2008. p. 153–77.