Journal of Southeast University (English Edition) Vol. 24, No. 2, pp. 168 – 173 June 2008 ISSN 1003—7985

Objective increment based metaheuristic for total flowtime minimization in no-wait flowshops

Zhu Xia Li Xiaoping Wang Qian

(School of Computer Science and Engineering, Southeast University, Nanjing 210096, China)

Abstract: To solve the NP-complete no-wait flowshop problems, objective increment properties are analyzed and proved for fundamental operations of heuristics. With these properties, whether a new generated schedule is better or worse than the original one is only evaluated by objective increments, instead of completely calculating objective values as the traditional algorithms do, so that the computational time can be considerably reduced. An objective increment-based hybrid genetic algorithm (IGA) is proposed by integrating the genetic algorithm (GA) with an improved various neighborhood search (VNS) as a local search. An initial solution generation heuristic (ISG) is constructed to generate one individual of the initial population. An expectation value-based selection mechanism and a crossover operator are introduced to the mating process. The IGA is compared with the traditional GA and two best-so-far algorithms for the considered problem on 110 benchmark instances. An experimental results show that the IGA outperforms the others in effectiveness although with a little more time consumption.

Key words: no-wait flowshops; total flowtime; objective increment; hybrid genetic algorithm

Scheduling plays a crucial role in manufacturing and service industries. It can be defined as setting the timetable for processing a given set of jobs on a set of machines in order to optimize a given measure of performance[1]. Among all the types of scheduling problems, no-wait flowshop problems (NWFP) have important applications in process-oriented enterprises[2] including chemical processing, food processing, etc. Modern manufacturing systems such as just-in-time (JIT) systems, flexible manufacturing environments and robotic cells can also be modeled as NWFP. In NWFP, the start of a job on a given machine should be delayed when it is necessary to ensure the completion time of the operation coinciding with the start of the next operation on the successive machine. In other words, the operations of a job have to be continuously processed. No interruption is permitted either on or between machines.

Total flowtime is an important performance measurement in practice. It equals the sum of completion times, the minimization of which may lead to stable or uniform utilization of resources, rapid turn-around of jobs, and minimizing in-process inventory[3]. NWFP with the objective of total flowtime minimization is usually denoted as $F_m \mid \mathrm{nwt} \mid \sum C_i$ [4]. Hall et al.[2,5] proved that $F_m \mid \mathrm{nwt} \mid \sum C_i$ are NP-complete in the strong sense even for the 2-machine case. Heuristics and metaheuristics are commonly used for NWFPs. Most metaheuristics can usually obtain better solutions than heuristics at the expense of much more CPU-time. However, heuristics need much less CPU-time but obtain worse solutions than metaheuristics.

For decades, many heuristics have been proposed for the problem considered in this paper. van Deman and Baker[6] proposed a branch and bound approach to find the optimal solution for special cases and introduced a set of procedures for generating lower bounds of optima. Adiri and Pohoryles[7] proved some properties of optimal schedules for 2-machine cases. Several theorems were also proved for polynomial bounded algorithms for m-machine problems with an increasing or decreasing series of dominating machines. van der Veen and van Dal[8] showed that the problem was solvable when the objective function was restricted to semi-ordered processing time matrices. Rajendran and Chaudhuri[9] presented two heuristic algorithms, RC1 and RC2, which distinctly outperformed the algorithms investigated by Bonney et al.[10-11]. Bertolissi[12] improved the EB algorithm[13] by job insertion methods. Experimental results show that the algorithm proposed by Bertolissi[12] is superior to RC1, RC2[9], EB[13] and the algorithm of Bonney and Gundry[10]. Chen et al.[14] developed a traditional genetic algorithm TGA. Experimental results show that the TGA yields better solutions in 168 times out of 200 than RC1 and RC2, but its performance becomes worse when the problem size increases. Fink and Voβ[15] examined the application of diverse metaheuristics on benchmark instances given by Taillard[16]. Results show that the reactive tabu search algorithm (SRTS) is the most effective. Aldowaisan and Allahverdi[17] constructed eight algorithms PH1 to PH4 and PH1p to PH4p and compared them with RC1 and the RC2[9], the algorithm of Bertolissi[12] and the TGA[14]. Results show that PH1p is the most effective. However, Ph1p[17] and SRTS[15] have not been compared so far.

In this paper, the objective increment-based IGA is proposed for $F_m \mid \mathrm{nwt} \mid \sum C_i$. The IGA works in a similar way to the HGA given by Ruiz et al.[18]. A heuristic ISG is constructed to generate one of the individuals in an initial population. An improved longest common subsequence (ILCS) based crossover operator is proposed based on the operator in Ref. [19] for the mating process. A selection mechanism for individual evolvement is improved by designing an expectation value on each individual in the population. The IGA is hybridized with a local search based on VNS[20]. Besides, objective increment methods are integrated with the IGA for CPU-time reduction.

Received 2008-02-18.

Biographies: Zhu Xia (1982—), female, graduate; Wang Qian (corresponding author), female, doctor, professor, qianwang6491@263.net.

Foundation items: The National Natural Science Foundation of China (No. 60504029, 60672092), the National High Technology Research and Development Program of China (863 Program) (No. 2008AA04Z103).

Citation: Zhu Xia, Li Xiaoping, Wang Qian. Objective increment based metaheuristic for total flowtime minimization in no-wait flowshops[J]. Journal of Southeast University (English Edition), 2008, 24(2): 168 – 173.

1 Problem Description

The NWFP is an important class of scheduling problems where n jobs $\{J_1, J_2, \ldots, J_n\}$ have to be scheduled on m consecutive machines $\{M_1, M_2, \ldots, M_m\}$ in the same order, and that once a job is started, it must be processed until completion without any interruption either on or between machines; i. e., waiting can occur only on the first machine. The following assumptions are taken into account in the NWFP:

1) Jobs consist of a strictly ordered sequence of operations and the sequence is predetermined.

2) Only one job can be processed on one machine at the same time.

3) Processing times of operations are deterministic and known in advance.

4) Once started on the first machine, the following operations of a job cannot be interrupted before completion, either on or between machines.

It is clear that a schedule is a job sequence derived from job set $\{J_1, J_2, \ldots, J_n\}$. For simplicity of analysis, a virtual job is introduced to a sequence as the first job denoted as $\pi_{[0]}$ of which the processing time on each machine is zero. A schedule π can be represented as $(\pi_{[0]}, \pi_{[1]}, \ldots, \pi_{[n]})$ where $\pi_{[i]} \in \{J_1, J_2, \ldots, J_n\} (1 \leqslant i \leqslant n)$ stands for the i-th job in π. Let $t_{k,i}$ be the processing time of $J_i (i = 0, 1, \ldots, n)$ on $M_k (k = 1, 2, \ldots, m)$, $D_{i,j}$ be the distance between the completion times of J_i and J_j on the last machine when J_j is directly processed after J_i. Thus, the distance can be computed by the following expression according to Ref. [21]:

$$D_{i,j} = \max_{k=1,2,\ldots,m} \left\{ \sum_{h=k}^{m} (t_{h,j} - t_{h,i}) + t_{k,i} \right\} \tag{1}$$

It is obvious that $D_{0,j} = \sum_{k=1}^{m} t_{k,j} (j = 1, 2, \ldots, n)$ and $D_{i,j} > 0$. Let $D^{\pi}_{[i],[j]}$ be the distance of job pair $(\pi_{[i]}, \pi_{[j]}) (0 \leqslant i, j \leqslant n, i \neq j)$, and let $d^{\pi}_{[i]}$ be the distance between $\pi_{[i]} (0 \leqslant i < n)$ and its direct successor $\pi_{[i+1]}$; the total flowtime of π can be defined as

$$F_n(\pi) = \sum_{i=0}^{n-1} (n - i) D^{\pi}_{[i],[i+1]} = \sum_{i=0}^{n-1} (n - i) d^{\pi}_{[i]} \tag{2}$$

Time complexities of $D_{i,j}$ in Eq. (1) and $F_n(\pi)$ in Eq. (2) are $O(m)$ and $O(mn)$, respectively. However, since $D_{i,j}$ only depends on $t_{k,i}$ which is predetermined, the distance between any of the two jobs maintains the same for any schedule. As a result, all the distances between any of the two jobs in set $\{J_1, J_2, \ldots, J_n\}$ may be calculated in advance and stored in matrix $\boldsymbol{M}$ as follows:

$$\boldsymbol{M} = \begin{bmatrix} \infty & D_{0,1} & D_{0,2} & \cdots & D_{0,n} \\ \infty & D_{1,1} & D_{1,2} & \cdots & D_{1,n} \\ \infty & D_{2,1} & D_{2,2} & \cdots & D_{2,n} \\ \vdots & \vdots & \vdots & & \vdots \\ \infty & D_{n,1} & D_{n,2} & \cdots & D_{n,n} \end{bmatrix}_{(n+1)\times(n+1)}$$

Hence, the complexity of Eq. (2) can be reduced to $O(n)$.

2 Objective Increment Method

For $F_m \mid \mathrm{nwt} \mid \sum C_i$, neighborhood construction in the searching processes of both heuristics and metaheuristics generally consist of three fundamental operators referred to insert, delete and swap. In terms of Eqs. (1) and (2), the total flowtime of a schedule is the weighted sum of adjacent distances (AD) and each AD depends only on the operation processing time of the two jobs. Therefore, a fundamental operator (insert, delete or swap) can only increase/decrease the objective values of the positions where the operator conducts in a schedule. In other words, ADs are identical before and after performing a fundamental operator for those positions the operator exerts no effect on. For the reason that only a few job positions change for fundamental operators and ADs in change positions can be obtained directly from $\boldsymbol{M}$, the corresponding objective increments can be calculated quickly.

Assume that π is a neighbor solution of π_0, then the objective value $F(\pi)$ can be calculated by adding increment $\Delta(\pi)$ (minus if π is better than π_0) caused by operators to $F(\pi_0)$ instead of being recalculated by Eq. (2), i. e. $F(\pi) = F(\pi_0) + \Delta(\pi)$. This is so-called objective increment method. For simplicity, $x_{i,j}(\pi)$, $s(k)$ and $y_{i,j}(\pi)$ are predefined as

$$x_{i,j}(\pi) = \begin{cases} 0 & i = n \\ (n - i)(D^{\pi}_{[i],[j]} - d^{\pi}_{[i]}) & \text{otherwise} \end{cases}$$

$$y_{i,j}(\pi) = \begin{cases} 0 & j = n \\ (n - j)(D_{i,\pi_{[j+1]}} - d^{\pi}_{[j]}) & \text{otherwise} \end{cases}$$

$$s(\pi) = \begin{cases} 0 & k < 0 \\ \sum_{i=0}^{k} d^{\pi}_{[i]} & k \geqslant 0 \end{cases}$$

Then different operators have different characteristics described as follows:

Theorem 1 If job J_q is inserted next to job $\pi_{[k]} (0 \leqslant k \leqslant n)$ in sequence $(\pi_{[0]}, \pi_{[1]}, \ldots, \pi_{[n]})$ of $F_m \mid \mathrm{nwt} \mid \sum C_i$, the objective increment would be $\Delta(n, k) = s(k-1) + (n - k + 1) D_{\pi_{[k]},q} + y_{q,k}(\pi)$.

Theorem 2 If job $\pi_{[k]} (0 \leqslant k \leqslant n)$ in sequence $(\pi_{[0]}, \pi_{[1]}, \ldots, \pi_{[n]})$ of $F_m \mid \mathrm{nwt} \mid \sum C_i$ is removed, the objective value would be reduced by $\partial(n, k) = s(k-1) + (n - k) \cdot d^{\pi}_{[k-1]} - y_{\pi_{[k-1]},k}(\pi)$.

Theorem 3 Exchange the position of job $\pi_{[i]}$ with job $\pi_{[j]}$ in sequence $(\pi_{[0]}, \pi_{[1]}, \ldots, \pi_{[n]})$ of $F_m \mid \mathrm{nwt} \mid \sum C_i$, the objective increment would be

$$\omega(i, j) = \begin{cases} x_{i-1,j}(\pi) + y_{\pi_{[i]},j}(\pi) + (n - i)(D^{\pi}_{[j],[i]} - d^{\pi}_{[i]}) & j = i + 1 \\ x_{i-1,j}(\pi) + y_{\pi_{[j]},i}(\pi) + x_{j-1,i}(\pi) + y_{\pi_{[j]},j}(\pi) & \text{otherwise} \end{cases}$$

All these three fundamental operators (characteristics) can be executed in time $O(1)$.

3 Proposed GA

3.1 Initial population

The population in this paper is composed of three parts: The first member is generated by the ISG; up to $B\%$ of the initial population is generated by the nearest neighborhood

(NN); the remaining $(100 - B)\%$ is filled with completely randomly generated sequences.

The first individual of the initial population is developed according to the characteristic of the problem considered. From Eq. (2), it can be seen that $F_n(\pi)$ is totally decided by the weighted sum of $d^{\pi}_{[i]}(0 \leqslant i \leqslant n-1)$ in $(\pi_{[0]}, \pi_{[1]}, \ldots, \pi_{[n]})$. Weight decreases as job position increases. Thus, in order to obtain the minimal $F_n(\pi)$, jobs with the smallest $d^{\pi}_{[i]}$ should be put as much in front as possible in a schedule while those with a large $d^{\pi}_{[i]}$ should be put backwards. It is actually to create a Hamilton route starting from $\pi_{[0]}$. Assigning $\boldsymbol{M}$ to $\boldsymbol{M}'$, a shortest Hamilton route is constructed through operations on $\boldsymbol{M}'$ in the ISG. The basic idea of the ISG is described as follows: Let μ_1 be null and μ_2 be the set of all jobs. To get rid of a circle, let $D_{j,\pi_{[i-1]}}$ be ∞ $(j = 0, 1, \ldots, n)$. Find the job with $\min\limits_{j=1,\ldots,n}\{D_{\pi_{[i-1]},j}\}$ from row $\pi_{[i-1]}$ in $\boldsymbol{M}'$ and let it be $\pi_{[i]}$, then $\mu_1 \leftarrow \mu_1 + \{\pi_{[i]}\}, \mu_2 \leftarrow \mu_2 - \{\pi_{[i]}\}$, $i \leftarrow i+1$. Do the step above repeatedly until $\mu_2 = \varnothing$ and then obtain a path $(\pi_{[0]}, \pi_{[1]}, \ldots, \pi_{[n]})$. Improve the solution with the RZ algorithm[22] and then obtain the initial solution π_0 with objective value $F_n(\pi_0)$ calculated by Eq. (2). The procedure of the ISG is described as follows:

1 $M' \leftarrow M$;
2 $\mu_1 \leftarrow \varnothing, \mu_2 \leftarrow \{\text{all jobs}\}, i \leftarrow 1$;
3 If $\mu_2 = \varnothing$, then go to 6;
4 For $j = 0$ to n
$D_{j,\pi_{[i-1]}} \leftarrow \infty$;
5 $D^{\pi}_{[i-1],[i]} \leftarrow \min\limits_{j=1,2,\ldots,n}\{D_{\pi_{[i-1]},j}\}, \mu_1 \leftarrow \mu_1 + \{\pi_{[i]}\}, \mu_2 \leftarrow \mu_2 - \{\pi_{[i]}\}, i \leftarrow i+1$, go to 3;
6 $\pi_0 \leftarrow \mathrm{RZ}(\pi_{[0]}, \pi_{[1]}, \ldots, \pi_{[n]}), F_n(\pi_0) \leftarrow \sum\limits_{i=0}^{n-1}(n-i) \cdot d^{\pi}_{[i]}$, stop.

The time complexity of the ISG is $O(n^3)$ by calculating F_n with Eq. (2), and is reduced to $O(n^2)$ using objective increment methods.

After selecting a job from $(\pi_{[0]}, \pi_{[1]}, \ldots, \pi_{[n]})$ as the first job, NN is applied by appending at each step an unscheduled job with a minimal inevitable delay to the last job of the partial sequence as yet uncompleted. In the IGA, NN is repeated for all the possible jobs in an identity permutation as the first job to construct $B\%$ of the initial population. The remaining $(100 - B)\%$ is filled with randomly generated sequences. By doing so, the diversity of the population is achieved.

3.2 Selection mechanism and generational scheme

The individuals presented in a population contain much information regarding the solution of the problem. For the selection of parents, each schedule (individual) has a total flowtime value that refers to the fitness value of that individual. The fitness value of a schedule i (denoted as Fit_i) is calculated from the fitness function defined as

$$\mathrm{Fit}_i = \max_{k=1,2,\ldots,N}\{\mathrm{TFT}_k\} - \mathrm{TFT}_i \tag{3}$$

where N is the size of the population and TFT_i is the total flowtime value of schedule i. Therefore, the procedure gives more chances to the ones with lower total flowtime values for being selected and generating offspring.

However, individuals of a population in the GA become more and more similar as each generation grows up. It destroys the population diversity and brings low searching efficiency. Thus, individual concentration C is introduced in this paper to define a measure of that diversity in terms of similarity degree R between two individuals in the current population. Entropy theory[23] is employed here to estimate the probability of the recurrence of patterns from an information source. The information entropy of locus x in the population is represented by $H_x(N) = -\sum\limits_{i=1}^{S} p_{ix}(x)\log p_{ix}$, where S is the variety of allele and p_{ix} means the probability that locus x is allele i. So, the average information entropy $H(N)$ is given as $H(N) = \frac{1}{M}\sum\limits_{x=1}^{M} H_x(N)$, where M is the size of genes in an individual. The similarity degree R_{vw} between individuals v and w can also be represented by $R_{vw} = \frac{1}{1+H(2)}$. For each individual of the population, the concentration C_v is calculated as $C_v = \frac{1}{N}\sum\limits_{w=1}^{N} Q_{vw}$, where $Q_{vw} = \begin{cases} 1 & R_{vw} \geqslant T \\ 0 & \text{otherwise} \end{cases}$, T is the predetermined threshold value. Therefore, the expectation value E of the individual v is considered as the judgment of the selection mechanism and calculated by

$$E_v = \frac{\mathrm{Fit}_v}{C_v} \tag{4}$$

Eq. (4) shows that the probability of being selected for the mating process is higher for individuals with a greater fitness value. It is lower for individuals with a higher concentration. At the beginning in the IGA, two individuals with the highest expectations are selected as parents for the mating process (selection_scheme 1). When there is no longer any improvement on the best solution for some successive generations, two individuals with the highest and lowest expectations, respectively, are selected instead (selection_scheme 2).

The generational scheme is a process by which new individuals in a new generation replace the worst members from the previous generation. The process is executed under the constraint that a new individual only replaces the worst one I_{worst} in the current population if its total flowtime is better than that of I_{worst} and the new sequence must be unique, i.e., the same sequence does not exist in the current population.

3.3 Crossover and mutation

ILCS is proposed as the crossover operator which is based on the operator in Ref. [19]. In ILCS, first, the longest common subsequence of two sequences (parent 1 and parent 2) is obtained with order of jobs which are expected to give good results, denoted as l. Copy jobs of l in the same positions as they are in the parents into offspring (child 1 and child 2). Treat elements not belonging to l in each parent as another sequence, and then repeat the same process as above

between the sequence and the current minimum individual. After that, another two longest common subsequences(l_1 and l_2) are obtained of which elements might be added into the offspring. By doing so, the locations of those jobs whose relative positions in both parents and the current minimum are the same are preserved. Then the remaining elements are swapped in the following way: The first element of parent 2 which is not presented in any longest common subsequence is copied at the first available place in child 1 and the element available at that position in parent 1 is copied at the first available position in child 2. For better understanding, an 11-job problem is illustrated in Fig. 1.

Parent 1:	4	6	9	3	7	2	10	8	1	5	11
Parent 2:	1	7	5	8	6	2	4	11	3	10	9
Opt:	4	2	5	8	3	9	11	1	7	6	10
Child 1:	4	7	5	3	6	2	10	8	1	11	9
Child 2:	1	7	9	2	6	8	4	5	3	10	11

Fig. 1 ILCS

In Fig. 1, l of parent 1 and parent 2 is $(4, 3, 10)$. Elements not belonging to l in parent 1 and parent 2 are $(6, 9, 7, 2, 8, 1, 5, 11)$ and $(1, 7, 5, 8, 6, 2, 11, 9)$. Opt is the current optimal individual in the population. The longest common subsequence between $(6, 9, 7, 2, 8, 1, 5, 11)$ and opt is $(2, 8, 1)$ (i. e. l_1), and that between $(1, 7, 5, 8, 6, 2, 11, 9)$ and opt is $(1, 7, 6)$ (i. e. l_2). After that, copy all the jobs in l, l_1 and l_2 in the same positions as they are in the parents. In parent 1, job 9 is the first element not belonging to any longest common subsequence. This goes into the first available slot in child 2 which is location 3. Likewise, job 7 from parent 2 goes into the first available slot in child 1 which is location 2. Repeat the above steps until two children are completed. The time complexity for finding the longest common subsequence is $O(n^2)$.

Mutation is done by selecting two different locations on an individual at random and inversing the order of jobs between them with a small probability P_m. It acts to enlarge searching space and to prevent the algorithm from a local optimum.

3.4 VNS local search

In general, VNS's main cycle consists of three phases: 1) Local search is to find local minimum in a neighborhood structure; 2) Move from one local search to another in case of no improvement for diversification of the search; 3) Shake means that a new solution generated by a local search is compared with the original one. If it is better, it replaces the original one and the algorithm starts again with the first local search. It is obvious that the performance of a local search depends on the choice of neighborhood structures denoted as $N_k(k = 1, 2, \ldots, k_{\max})$, where $k_{\max}$ is the maximal number of neighborhood structures. In this paper, VNS is embedded in the IGA but only two neighborhood structures DDI and SDI (i. e. $k_{\max} = 2$) are used:

1) DDI: d jobs deleted and inserted

d jobs are randomly removed from sequence π and permutated in reverse order of their positions in π. These d jobs form sequence π_d and the rest jobs in π form sequence π_r. Each job in π_d is removed and inserted into the position with minimal objective increments until π_d is empty. Thus, a complete candidate solution is yielded.

2) SDI: single job deleted and inserted

One job is randomly removed from sequence π and reinserted into the position with minimal objective increments.

Let N_1 and N_2 be neighborhood structures DDI and SDI respectively, and $N_k(x)$ be a set of solutions generated in the k-th neighborhood of sequence x. The procedure of VNS to sequence π is described as follows:

```
VNS(π)
{s0←π, k←1;
    Choose Nk, k = 1, 2;
s←Perturbation(s0);
Do {
    s* ←ChooseBestOf(Nk(s));
    If F(s*) < F(s), then s←s*, k←1;
    else k←k + 1;
} While(k≤2)
If F(s) < F(π), then π←s; }
```

In this VNS algorithm, π is assigned to the incumbent solution s_0. After choosing the neighborhood structure N_k, s_0 is disturbed to avoid getting trapped into local minima. Then the local search is applied to the perturbed solution s. During the local search, DDI/SDI is applied to s repeatedly to generate various neighbor solutions, and the best of those solutions is selected as s^* for further evaluation. VNS is applied on individuals after selection, crossover or mutation with a probability P_{vns}. Besides, at the end of each generation, the best individual of the current population is also enhanced by VNS with probability $2P_{\text{vns}}$.

4 IGA

It is the fact that, at some given time the population can achieve a sufficiently low diversity for the process to stall around a local optimum. To overcome this problem, a restart mechanism is adopted as follows: Find $W\%$ individuals of the population with high individual concentration and replace them with new individuals generated at random. The algorithm stops when there is no improvement to the current optimal solution after three times of restart. The whole procedure of the IGA can be described as follows:

```
gen←0, count←0, time←0, flag←true;
POP←initialization(), result←minTFT(POP);
Do {
  //evaluate individual
      CaculateExpectationValue(POP);
  //selection mechanism
      If(flag = true)
      //selection_scheme1
      p1, p2←IndividualWithMaxExpectValue(POP);
  else {
      //selection_scheme2
      p1←IndividualWithMaxExpectValue(POP);
      p2←IndividualWithMinExpectValue(POP); }
```

```
  //crossover
      (c1, c2) ←ILCS(p1, p2, result);
//mutation
    c1←mutation(c1, P_m), c2←mutation(c2, P_m);
//local search
    c1←VNS(c1, P_vns), c2←VNS(c2, P_vns);
//generational scheme(repeat this procedure for c2)
    I_worst←max TFT(POP);
    If ((TFT(c1) < TFT(I_worst)) and (
        (TFT(c1) ≠ TFT(pop)) or(c1 unique)))
          I_worst←c1;
    gen← gen + 2;
    if(gen = N)
    {
      I_best←min TFT(POP),
      I_best←VNS(I_best, 2P_vns);
      If(TFT(I_best) < TFT(result))
      {
          result←I_best;
          time←0, count←0, flag←true; }
      else {
          time + +, count + +; }
      gen←0;
    }
//selection judgment
    If(time ≥ TIME)
        flag←false;
//restart scheme
    If(count ≥ COUNT)
    {
        POP ←Restart(POP);
        time←0, count←0, flag←true; }
} While(termination criterion is not satisfied)
Return result, the algorithm stops.
```

5 Experimental Results

The proposed IGA is compared with the traditional GA and the two best existing algorithms SRTS[15] and PH1p[17] on benchmark instances of Taillard[16]. For better analysis, 110 benchmark instances are divided according to the problem scales, which means instances with the same scales are put together as a group. Hence, there are 11 groups, each of which consists of 10 instances. The main parameters in the IGA are set as follows: $N = 50$, $B = 40$, $W = 30$, TIME = 8 and COUNT = 12. All the algorithms are implemented by Visual C++ and performed on PC 2.93 GHz with 512 MB RAM. For each instance, 5 runs are conducted.

To compare the performance of the three heuristics in effectiveness, two measurements are analyzed: average relative percentage deviation(ARPD) and CPU time. Fig. 2 and Fig. 3 show the ARPDs and CPU times of the four algorithms in 11 benchmark groups.

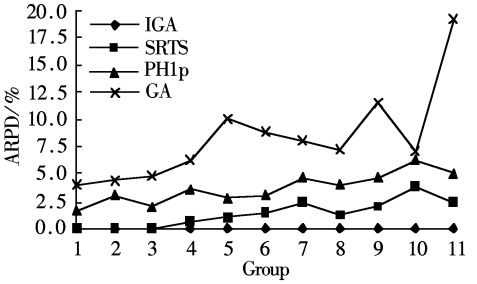

Fig. 2 Comparison of ARPD

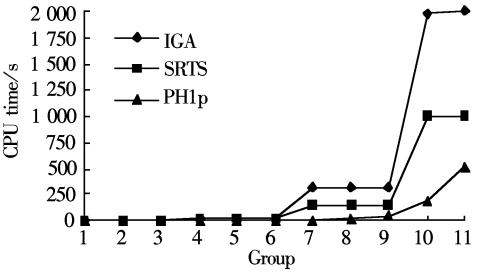

Fig. 3 Comparison of CPU-time

From Fig. 2, it can be seen that the ARPD of the IGA remains at zero for all 11 groups. As the problem size increases, ARPDs of SRTS and PH1p increase with fluctuation and reach the tops at group 10 with 3.933% and 6.214%. The average ARPDs of SRTS and PH1p are 1.365% and 3.701%. However, the ARPD of the GA fluctuates greatly as the problem size increases. The ARPD value of the GA in each group is much larger than those of the other three and it even reaches 20% at group 11. The average ARPD of the GA is 8.313%. Therefore, the proposed IGA seems to be the most effective so far.

The CPU time of the GA is not illustrated in Fig. 3 for the reason that it is much more than the times of the other three and not even of the same quantity class. For example, it needs 23 886 s at group 11 in the GA and it is about twelve times larger than that of the proposed IGA. Fig. 3 shows that, although the operation steps of the IGA are much more than those of SRTS, the CPU time of the IGA is only about twice that of SRTS. This may due to the fact that time is largely reduced by the objective increment methods which obtain the objective value only by calculating the variables of the objective value instead of complete recalculation. Besides, the IGA outperforms PH1p on the effectiveness at a little cost of efficiency. Therefore, the IGA is promising and worthy in practical projects.

6 Conclusion

In this paper, no-wait flowshops with total flowtime minimization is considered. According to the problem characteristics, the independency of time distances between jobs in schedule is analyzed and objective increment properties of fundamental operators are described. The IGA is proposed according to the obtained properties. In the IGA, the ISG is constructed to generate one individual of an initial population. The selection mechanism of individual evolvement is improved by evaluating the expectation value of each individual. The ILCS crossover operator is developed for the mating process. The IGA is incorporated with a VNS-based local search where DDI and SDI are constructed as two kinds of neighborhood structures. In addition, a restart mechanism is adopted to prevent the algorithm from a local optimum. Finally, the computational results are given and demonstrate the superiority of the IGA to the traditional GA and the two best existing algorithms on effectiveness at the cost of a little more CPU-time.

References

[1] Pinedo M. *Scheduling*: *theory*, *algorithm*, *and systems* [M]. Englewood Chills, NJ: Prentice-Hall, 1995.

[2] Hall N G, Sriskandarajah C. A survey of machine scheduling problems with blocking and no-wait in process [J]. *Operations Research*, 1996, **44**(3): 510 − 525.

[3] Rajendran C. A no-wait flowshop scheduling heuristic to minimize makespan[J]. *Journal of the Operational Research Society*, 1994, **45**(4): 472 − 478.

[4] Wang L, Zheng D Z. An effective hybrid optimization strategy for job-shop scheduling problem [J]. *Computer and Operation Research*, 2001, **28**(6): 585 − 596.

[5] MacCarthy B L, Liu J. Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling [J]. *International Journal of Production Research*, 1993, **31**(1): 59 − 79.

[6] van Deman J M, Baker K R. Minimizing mean flow time in the flowshop with no intermediate queues [J]. *AIIE Transactions*, 1974, **6**(1): 28 − 34.

[7] Adiri I, Pohoryles D. Flowshop/no-idle or no-wait scheduling to minimize the sum of completion times [J]. *Navel Research Logistics Quarterly*, 1982, **29**(3): 495 − 504.

[8] van der Veen J A A, van Dal R. Solvable cases of the no-wait flowshop scheduling problem [J]. *Journal of the Operational Research Society*, 1991, **42**(11): 971 − 980.

[9] Rajendran C, Chaudhuri D. Heuristic algorithms for continuous flow-shop problem [J]. *Naval Research Logistics*, 1990, **37**(5): 695 − 705.

[10] Bonney M C, Gundry S W. Solutions to the constrained flowshop sequencing problem [J]. *Operational Research Quart*, 1976, **27**(4): 869 − 883.

[11] King J R, Spachis A S. Heuristics for flow-shop scheduling [J]. *International Journal of Production Research*, 1980, **18**(3): 343 − 357.

[12] Bertolissi Edy. Heuristic algorithm for scheduling in the no-wait flow-shop [J]. *Journal of Materials Processing Technology*, 2000, **107**(1/2/3): 459 − 465.

[13] Bertolissi Edy. A simple no-wait flow-shop scheduling heuristic for the no-wait flowshop problem[C]//*Proceedings of the* 15*th International Conference on Computer-Aided Production Engineering*, *CAPE'*99. Durham, UK, 1999.

[14] Chen C L, Neppalli R V, Aljaber N. Generic algorithms applied to the continuous flow shop problem [J]. *Computers and Industrial Engineering*, 1996, **30**(4): 919 − 929.

[15] Fink A, Voβ S. Solving the continuous flow-shop scheduling heuristic to minimize makespan[J]. *Journal of the Operational Research*, 2003, **151**(3): 400 − 414.

[16] Taillard E. Benchmarks for basic scheduling problems [J]. *European Journal of Operational Research*, 1993, **64**(2): 278 − 285.

[17] Aldowaisan T, Allahverdi A. New heuristics for *m*-machine no-wait flowshop to minimize total completion time [J]. *OMEGA*, 2004, **32**(5): 345 − 352.

[18] Ruiz Rubén, Maroto Concepción, Alcaraz Javier. Two new robust algorithms for the flowshop scheduling problem [J]. *OMEGA*, 2006, **34**(5): 461 − 476.

[19] Iyer Srikanth K, Saxena Barkha. Improved genetic algorithm for the permutation flowshop scheduling problem [J]. *Computers and Operations Research*, 2004, **31**(4): 593 − 606.

[20] Blum Christian, Andrea Roli. Metaheuristics in combinatorial optimization: overview and conceptual comparison [J]. *ACM Computing Surveys*, 2003, **35**(3): 268 − 308.

[21] Li Xiaoping. Heuristic for no-wait flow shops with makespan minimization [J]. *International Journal of Production Research*, 2008, **46**(9): 2519 − 2530.

[22] Rajendran C, Ziegler H. An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs [J]. *European Journal of Operational Research*, 1997, **103**(1): 129 − 138.

[23] Abramson N. *Information theory and coding* [M]. New York: McGraw-Hill, 1963: 129.

基于目标增量的最小化总完工时间无等待流水调度智能算法

朱 夏 李小平 王 茜

(东南大学计算机科学与工程学院,南京 210096)

摘要:针对 NP-完全的无等待流水作业调度问题,改变传统求解调度序列目标函数的模式,分析并证明启发式算法基本算子的目标增量性质,通过目标函数变化量判断新解的优劣,大大降低算法所需计算时间. 提出将变化邻域搜索(VNS)作为一种局部搜索机制混合入遗传算法的智能算法 IGA 求解所考虑的问题,根据问题特点构造 ISG 算法产生初始种群中的一个个体,设计基于期望值的个体选择机制和进化过程交叉算子 ILCS. 采用 110 个经典 Benchmark 实例,将所提出的 IGA 算法与传统遗传算法以及求解该问题目前最好的 2 种算法进行比较,实验结果表明 IGA 算法在略有耗时的情况下,性能上明显优于其他 3 种算法.

关键词:无等待流水调度;总完工时间;目标增量;混合遗传算法

中图分类号:TP278