Int J Adv Manuf Technol (2006) 31: 350–359
DOI 10.1007/s00170-005-0194-7

ORIGINAL ARTICLE

Fuh-Der Chou · Pei-Chann Chang · Hui-Mei Wang

A hybrid genetic algorithm to minimize makespan for the single batch machine dynamic scheduling problem

Received: 1 November 2004 / Accepted: 6 June 2005 / Published online: 24 November 2005

Abstract This paper considers a single batch machine dynamic scheduling problem, which is readily found in the burn-in operation of semiconductor manufacturing. The batch machine can process several jobs as a batch simultaneously, within the capacity limit of the machine, and the processing time is represented by the longest processing time among all jobs in a batch. For a single batch machine problem with arbitrary job release time, we proposed an improved algorithm (merge-split procedure) to refine the solution obtained by the *LPT-BFF* heuristic, and two versions of a hybrid genetic algorithm (GA) are introduced in this paper. Each version of the hybrid GA diversifies job sequences using the GA operators in stage 1, forms batches in stage 2, and finally sequence the batches in stage 3. The difference is that merge-split procedures are involved in the second version of the hybrid GA. Computational experiments showed that the hybrid GA would obtain satisfactory average solution quality and the merge-split procedures would be good at reinforcing the solution consistency of the hybrid GA.

F.-D. Chou
Department of Industrial Engineering & Management,
Ching Yun University,
Chungli, Tao Yuan,
Taiwan

P.-C. Chang
Department of Industrial Engineering and Management,
Yuan-Ze University,
Nei-Li, Tao Yuan,
Taiwan

H.-M. Wang (✉)
Department of Industrial Management,
No. 1, Van-Nung Rd, Chung-Li,
Taoyuan, 32045, Taiwan, People's Republic of China
e-mail: amywang@vnu.edu.tw
Tel.: +886-3-4515811
Fax: +886-3-4632679

Keywords Batch machine · Genetic algorithm (GA) · Makespan · Scheduling

1 Introduction

The single batch machine problem is a scheduling problem where jobs can be grouped into batches which are within the capacity limit of a machine conforming to the following requirements: the processing time of a batch is equal to the longest processing time among all jobs in the batch; once a batch is processed, it cannot be interrupted; and no jobs can be removed from the machine until the process is completed. This problem is motivated by the burn-in operation in semiconductor final testing. Early Ikura and Gimple [1] studied the batch machine problem, proposing an optimal algorithm for the single batch machine problem with identical job processing times, unit job sizes, dynamic job arrivals, and an objective of minimizing makespan. Lee et al. [2] gave a detailed description for burn-in operation. Following their study of the problems that batch machines encountered in a number of different production environments including diffusion and burn-in operations in semiconductor fabrication, heat treatment operations in metalworking, batch machine problems have been extensively treated in the literature. For the batch machine problem where capacity of a machine is measured by the number of jobs it could accommodate, Lee et al. [2] studied the problem in which the processing time of jobs are the same and the relationship of release time (r_i) and due date (d_i) is agreeable, i.e., $r_i \leq r_j$ implying $d_i \leq d_j$. They proposed dynamic programming algorithms to minimize maximum tardiness ($T_{\max}$) and the number of tardy jobs $\left(\sum_i U_i\right)$. Li and Lee [3] showed that there is an optimal solution in which the jobs be arranged in such a way that $r_1 \leq r_2 \leq ... \leq r_n$, $d_1 \leq d_2 \leq ... \leq d_n$ and $p_1 \leq p_2 \leq ... \leq p_n$, and applied dynamic programming algorithms to minimize maximum tardiness

and the number of tardy jobs. Sung and Kim [4] considered the two batch machine flow shop problem where the processing time of a batch depends on an individual machine. The problem is an example of steps in the wafer fabrication process such as oxidation, etching operations. For this problem they presented dynamic programming algorithms to minimize maximum tardiness, number of tardy jobs, and total tardiness.

Most of the algorithms mentioned above consist of employing a sorting rule to determine a job sequence, and then apply dynamic programming or other methods to determine the batches. This type of algorithm is also

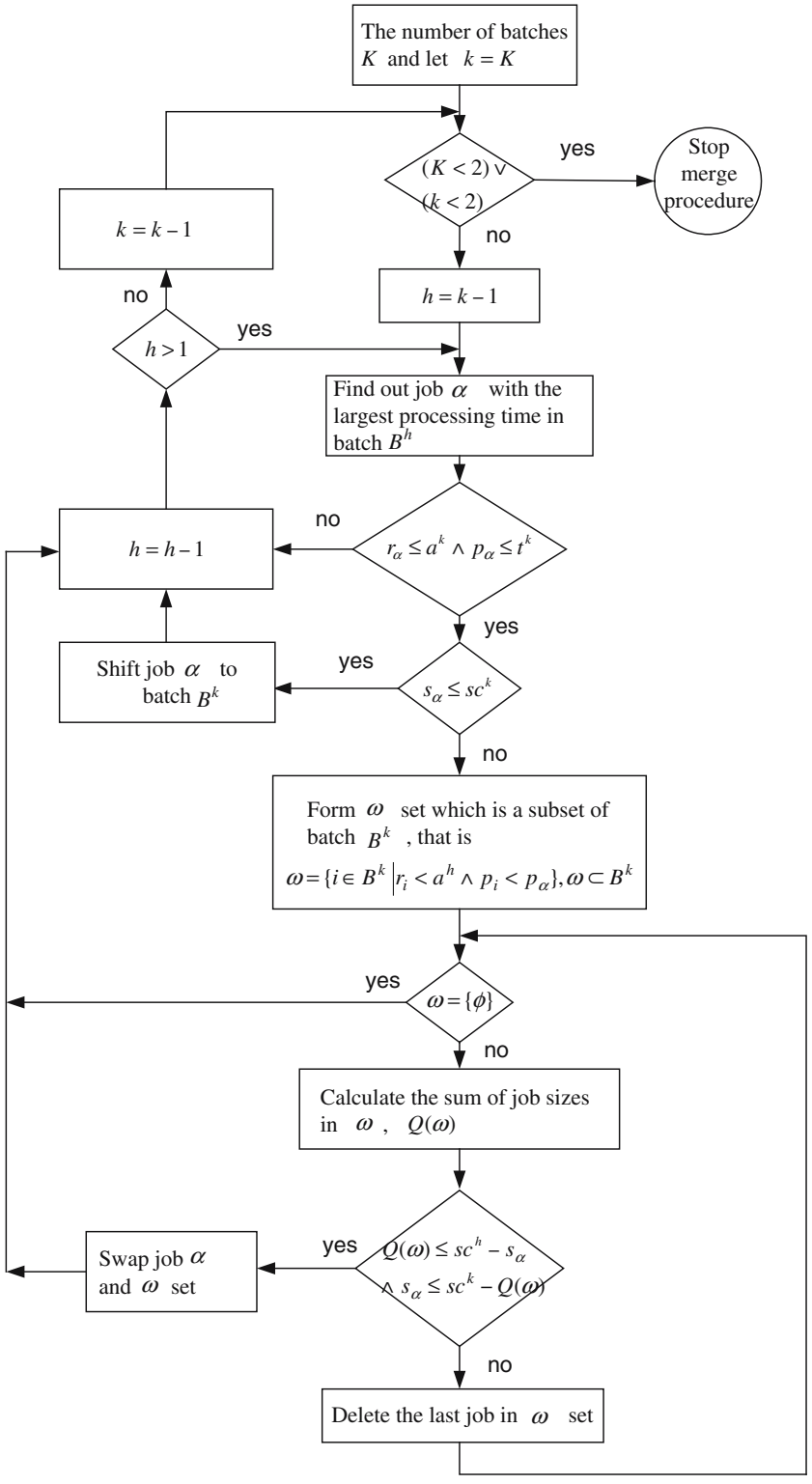

Fig. 1 Flowchart of merge procedure

applied to other criteria. To minimize total completion time, Chandru et al. [5] indicated jobs are indexed according to the shortest processing time (*SPT*), and there is an optimal schedule where all batches are composed of consecutively indexed jobs. According to this property, Chandru et al. [5] and Dupont and Ghazvini [6] used a branch and bound method for a static version of the single batch machine problem. Hochbaum and Landy [7] applied this property to the dynamic programming algorithm in minimizing total flowtime for an m-type burn-in problem in which jobs belonging to type t have the same processing time p_t. In contrast, the longest processing time (*LPT*) dispatch rule commonly determines the sequence of jobs for makespan criterion, and it can be found in the *FBLPT* algorithm proposed by Bartholdi [8]. Subsequently, Dupont and Dhaenens-Flipo [9] presented an easy algorithm similar to the *FBLPT* algorithm to obtain an optimal solution for the special case of a single batch machine problem where jobs have identical capacity requirements. In addition, they provided a branch and bound method for the problem with non-identical job sizes and identical release times. Uzsoy [10] proposed five heuristics to minimize the makespan for the static version of batch machine problem with non-identical job sizes. These five heuristics applied different sorting rules to the job sequence decision, and the results revealed that the *LPT* dispatch rule was better than other rules when the batching rule was the Batch First-Fit (*BFF*) heuristic. Following that, Zhang et al. [11] provided worst-case analysis for Uzsoy's proposed heuristics.

When release times of jobs are arbitrary, it is difficult to decide which kinds of sorting rules are dominant to determine the sequence of jobs. Lee and Uzsoy [8] and Sung and Choung [12] applied dynamic programming algorithms for grouping jobs as batches under the assumption that job sequence is predetermined for the single batch machine problem. Since the job sequence would result in a significant influence on solution quality for the dynamic version of the single batch machine problem, to date there has been no dominant sorting rule of the job sequence for this problem. Therefore this paper applies a genetic algorithm to generate the job sequence pool in order to decrease the influence of job sequence on solution quality. We also proposed a novel procedure to improve the solution quality under the assumption that batches are given. Moreover, a variety of experimental data were used to compare these proposed algorithms with respect to solution quality and execution time.

This paper considers the single batch machine problem in which release times of jobs are arbitrary, job sizes are non-identical, and the objective function is to minimize the makespan. The definitions of the problem and notations are described in Section 2. Sections 3 and 4 present the proposed novel procedure and the genetic algorithm implementation. In Section 5, experimental results are presented that examine the effectiveness of the proposed approaches. Finally, Section 6 offers brief concluding comments.

2 Minimizing the makespan on a single batch machine

This paper considers the burn-in oven scheduling problem where burn-in ovens are modeled as batch machines. For

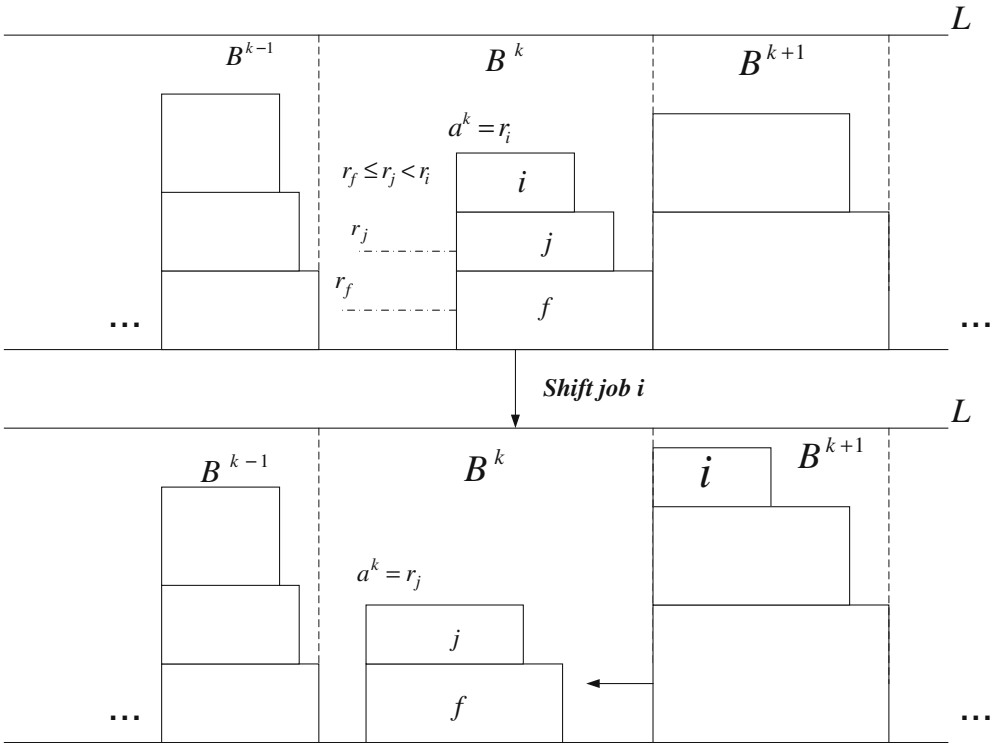

Fig. 2 Shift process of the split procedure

the single batch machine dynamic scheduling problem, the following assumptions and notations are described:

(1) There are n jobs to be processed. The processing time and the release time of job i are denoted by p_i and r_i, respectively.
(2) All jobs are compatible, and job sizes (s_i) of job i are less than or equal to maximum capacity L of machine. All data of p_i, r_i, s_i and L are deterministic and known a priori.
(3) The sum of job sizes of jobs in a batch must be less than or equal to the maximum capacity of the machine, i.e., $bs^k \leq L$ where $bs^k = \sum_{i \in B^k} s_i$.
(4) Once processing of a batch has begun, other jobs cannot be added or removed from the machine until processing is finished. The processing time of batch B^k is given by the longest processing time of the jobs in the batch, i.e., $t^k = \max\{p_i | i \in B^k\}$, the available time

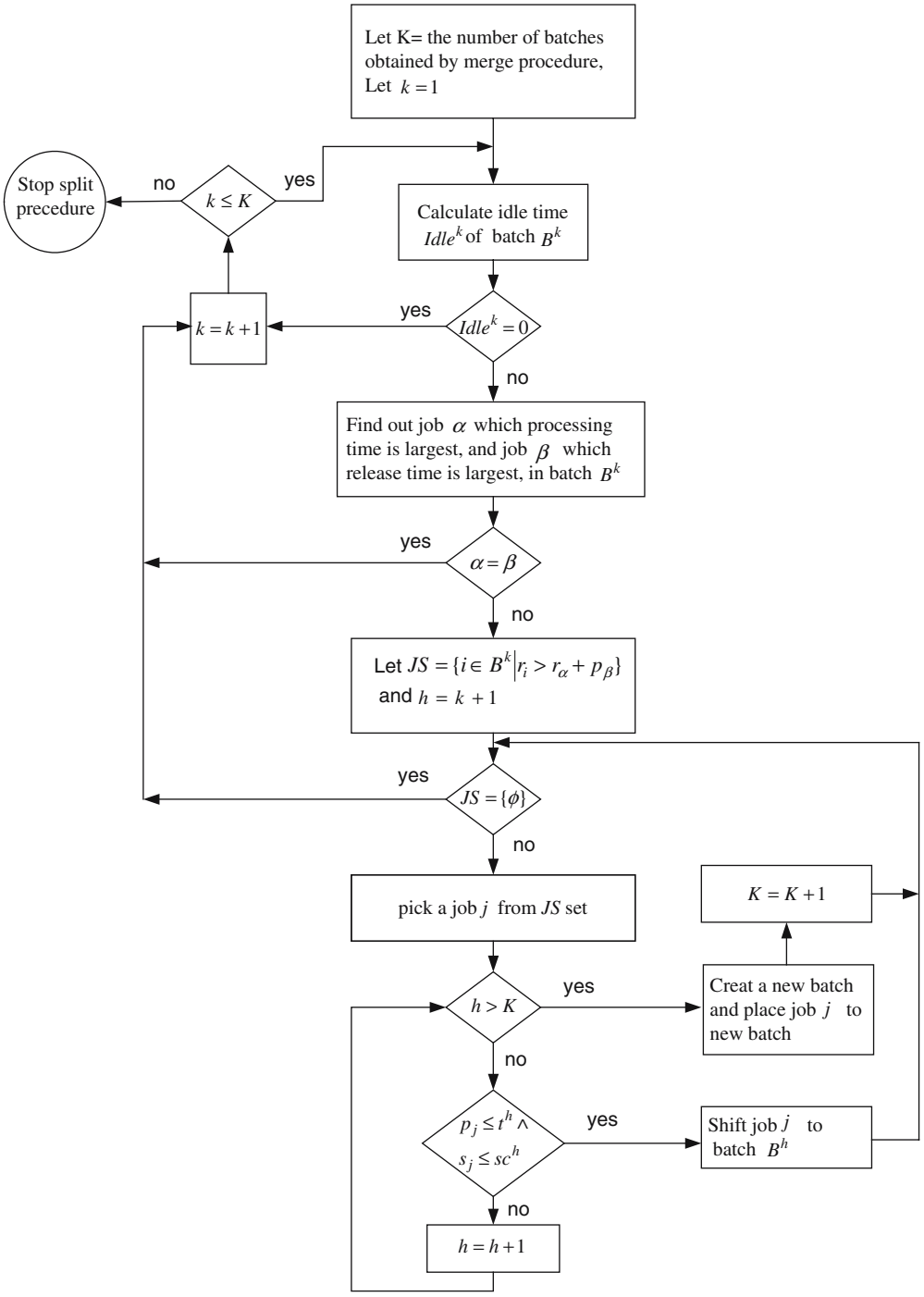

Fig. 3 Flowchart of the split procedure

and completion time of batch B^k are denoted by a_k where $a^k = \max\{r_i | i \in B^k\}$ and cB^k where $cB^k = \max(cB^{k-1}, a^k) + t^k$, respectively.
(5) Machine breakdown is not allowed.
(6) The objective criterion is to minimize the makespan, $C_{\max}$.

3 Merge-split procedures

For minimizing the makespan on a single batch machine problem where all release times are zero and job sizes are non-identical, Dupont and Dhaenens-Flipo [9] proposed a definition of *Left-Shifted* schedule if no job i in the batch B^h satisfied $s_i \leq sc^k$ and $p_i \leq t^k$, where sc^k is leftover capacity of batch B^k, batch B^k is processed before batch B^h, and the starting time of batch B^{k+1} is equal to the completion time of batch B^k. They proved *left-shifted* schedules are dominant for any regular criteria. According to the definition of *left-shifted* schedule, we can continue to shift job i in batch B^h which satisfies $s_i \leq sc^k$ and $p_i \leq t^k$ from B^h to B^k until the schedule becomes a *left-shifted* schedule. This procedure is clearly an improvement, and we here call it the *left-shifted* procedure. Based on the *left-shifted* procedure, we developed a more generalized improvement procedure, called the merge procedure, for a single batch machine problem with arbitrary job release time. The merge procedure consists of right-shifted and swapped improvement. The right-shifted improvement is to shift job i from batch B^h to batch B^k if the following conditions are satisfied.

(1) Batch B^h preceded batch B^k,
(2) Job i is a longest job, i.e., its processing time is equal to the processing time t^h of batch B^h.
(3) $r_i \leq a^k$, $p_i \leq t^k$ and $s_i \leq sc^k$.

Swapped improvement makes an exchange for job i and ω set where the member of jobs in ω set is greater than or equal to one. The swapped improvement is triggered when the following conditions are satisfied.

(1) Batch B^h preceded batch B^k.
(2) Job i is a longest job, i.e., its processing time is equal to the processing time t^h of batch B^h, and $p_i \leq t_k$.
(3) ω set is a subset of batch B^k, and the processing times of all jobs belonging to ω set are less than the processing time p^i of job i.
(4) The job sizes of job i are less than or equal to the leftover capacity of batch B^k plus the sum of job sizes of ω set, i.e., $s_i \leq sc^k + \sum_{j \in \omega} s_j$.
(5) The sum of job sizes of jobs belonging to ω set is less than or equal to the leftover capacity of batch B^h plus job sizes of job i, i.e., $\sum_{j \in \omega} s_j \leq sc^h + s_i$.

The merge procedure is analogous to the forward procedure which progresses from the batch in position k toward the batch in position 1 when a batch schedule is given. A flowchart describing the merge procedure is shown in Fig. 1.

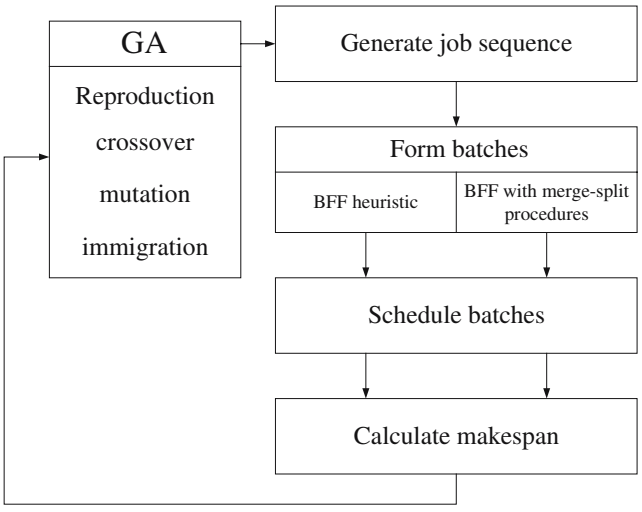

Fig. 4 Hybrid genetic algorithm

Now consider the case where the machine's capacity is larger than the sum of all job sizes, i.e., $L > \sum_{i=1}^{n} s_i$. When all release times of jobs are the same, placing all jobs in a batch is optimal for the makespan criterion. However this is not always true for cases where jobs have different release times. For example, assume there are five jobs where release time, processing time and job sizes are {10,48,28,3,48}, {35,16,30,38,9} and {9,1,1,8,10}, respectively, and the machine's capacity is 40. We place all jobs in one batch because the machine has enough space to accommodate all jobs, and the makespan is 86. However, if we place jobs 1, 3, 4 in the first batch, and jobs 2, 5 in the second batch, then the makespan would decrease to 78. Therefore, for a dynamic version of the single batch machine problem, it is necessary to decide whether to stop current batching and then continue next batching or to wait for the arrival of another job, even when the current batch has still enough space to accommodate other jobs. For this example we can conclude that the *BFF* heuristic proposed by Uzsoy [10] to generate batches would yield the same result, no matter what job order was used. The *BFF* heuristic would cause unnecessary idle time due to waiting for the arrival of another job when the current batch has enough space to accommodate it. To avoid this situation,

Table 1 Experimental design of a preliminary test

Parameter	Value
Population size	200
(Crossover, immigration, reproduction) (%)	(50, 45, 5), (70, 25, 5), (90, 5, 5), (98, 1, 1)
Mutation (%)	0.25, 1.25, 2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20

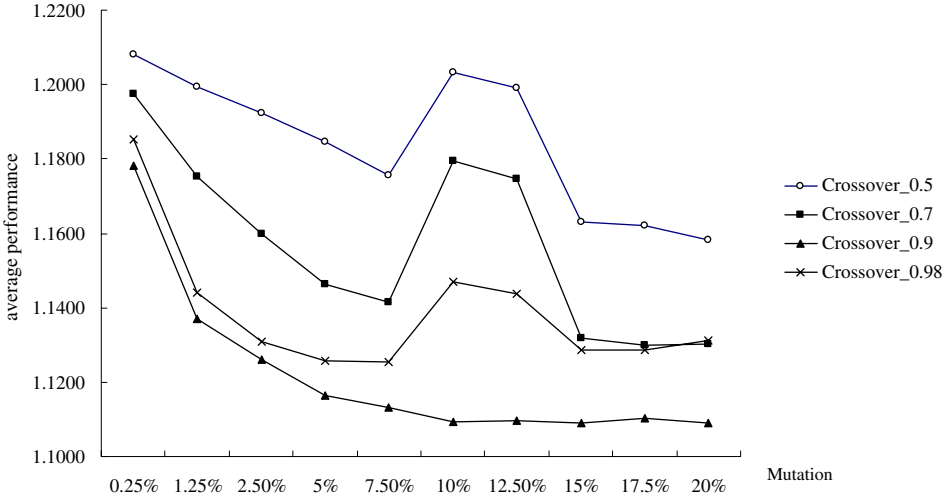

Fig. 5 Crossover-mutation plot for experimental design (Table 1)

we proposed a split procedure to decrease idle time of batches. This procedure starts with batch B^k with nonzero idle time, i.e., $cB^{k-1} < a^k$, and shift job i that satisfied conditions $r_i = a^k$ and $p_i < t^k$ from batch B^k to subsequent batch B^{k+j}, $j \geq 1$, which has enough space to accommodate job i and the processing time of the batch is larger than p_i or it creates a new batch to place the job i.

The shift process could reduce the idle time of batch B^k, and it is possible that the makespan would be decreased (see Fig. 2). Assuming that a batch schedule is given, the split procedure proceeds from the batch in position 1 toward the batch in position k, and the flowchart of the split procedure is shown in Fig. 3.

Merge-split procedures start with a given batch schedule obtained by any heuristic, followed by a merge procedure and a split procedure that together have the possibility of improving the performance of an initial batch schedule.

4 Hybrid genetic algorithm

GAs have been applied extensively to combinatorial optimization problems, including scheduling problems. In GA applications for single machine scheduling problems, the sequences of jobs are considered as individuals (chromosomes) of a population, and the objective function of an individual is measured as the fitness value. An individual with a higher fitness value has a larger probability to be selected for reproduction, and crossover/mutation for next generation.

Table 2 Parameter settings for the hybrid GA

Parameter	Value
Population size (*ps*)	200
Reproduction	5%
Crossover probability	90%
Mutation probability	10%
Immigration	5%

This paper proposes two versions of a hybrid genetic algorithm with three stages. Stage 1 is to generate a variety of job sequences using the GA operators, forming the batches in stage 2 and scheduling the batches on a single batch machine in stage 3. In stage 2, the first version of hybrid GA uses only the *BFF* heuristic to form batches, while merge-split procedures are applied to improve the result of the *BFF* heuristic in the second version. Because if batches are fixed and known in advance, Sung et al. showed that scheduling the batches by earliest release time (*ERT*) order would be optimal [13]. Therefore, when the batches have been determined by stage 2, they are assigned on a single batch machine by non-decreasing order of batches' available time in stage 3. The stages of this hybrid genetic algorithm are shown in Fig. 4.

The GA is principally used to decide the sequence of jobs in this paper, and the operators of the GA are described below.

Population generation A job sequence is represented as chromosome whose length is equal to the number of jobs to be scheduled. The initial population includes chromosomes obtained by first in, first out (*FIFO*), *LPT* and *SPT* dispatch rules. The rest of the chromosomes were generated randomly. The population size for each generation is 200. The next generation is from the combination of reproduction, crossover/mutation, and immigration. For example, 10% of the next generation comes from the top 10% of solutions from the last generation, 80% from the crossover/mutation, and 10% is randomly generated.

Crossover A one-point crossover [14] is used in the GA, in which two parent chromosomes are selected randomly

Table 3 Representation of three approaches

Approach	Notation
A1	*H(LPT,BFT+merge_split,ERT)*
A2(version 1 of hybrid GA)	*H(GA,BFF,ERT)*
A3(version 2 of hybrid GA)	*H(GA,BFF+merge_split,ERT)*

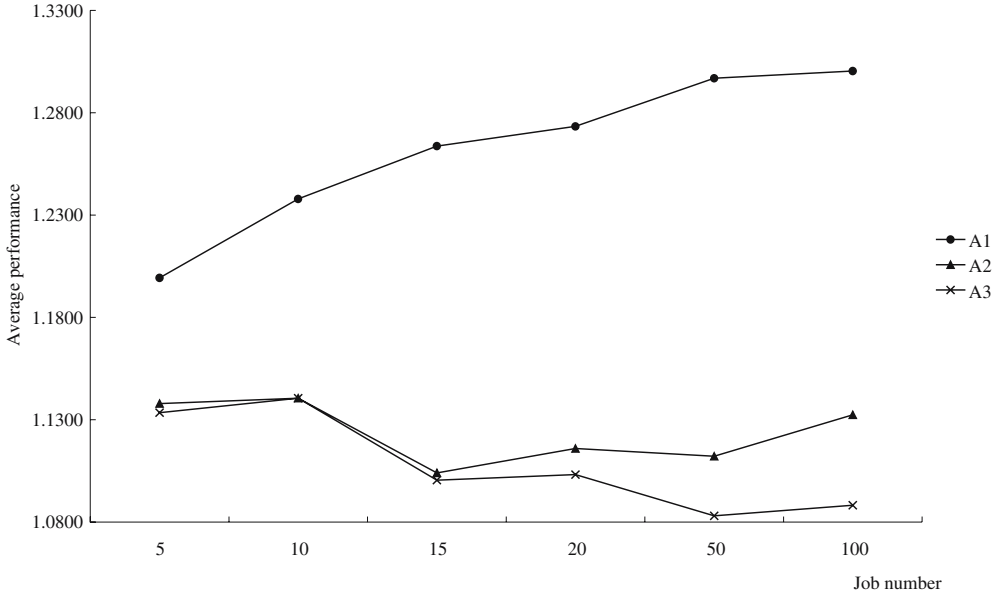

Fig. 6 Average performance for small job sizes

according to fitness, and the crossover point is randomly chosen.

Mutation Swapping and shift mechanisms are commonly used in the mutation operator. Here, we used a swapping mutation which randomly selects genes of offspring to change with a predefined probability.

Selection Parts of chromosomes are selected from the population for crossover using the roulette wheel technique [14]. Because the makespan is minimized, we need to transform the objective function for maximization. We defined $fitness'(\sigma_i) = 1.10 \cdot Max(sol(\sigma_i), \ldots, sol(\sigma_{2n})) - sol(\sigma_i)$, where $sol(\sigma_i)$ is the makespan value of σ_i, and σ_i means the batch schedule obtained through stage 2 (form batches) and stage 3 (schedule batches) for the *i*th job sequence. The probability of selection of a certain chromosome is calculated by $\frac{fitness'(\sigma_i)}{\sum_{i \in \psi} fitness'(\sigma_i)}$, where ψ is the current population.

Stopping criterion In this paper three stopping criterion are used to control the hybrid GA: (1) when 1000 generations have been reached, i.e., *maxg*=1000, (2) there has been no improvement for the best solution in the last 100 generations, and (3) the best solution is equal to the lower bound. If any of these three stopping conditions is satisfied, the GA is stopped.

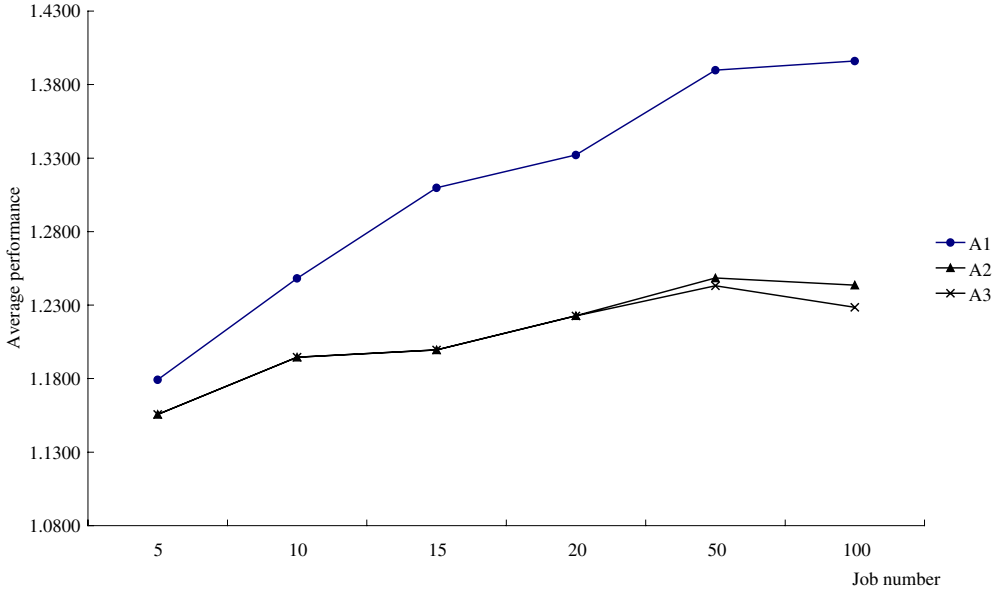

Fig. 7 Average performance for large job sizes

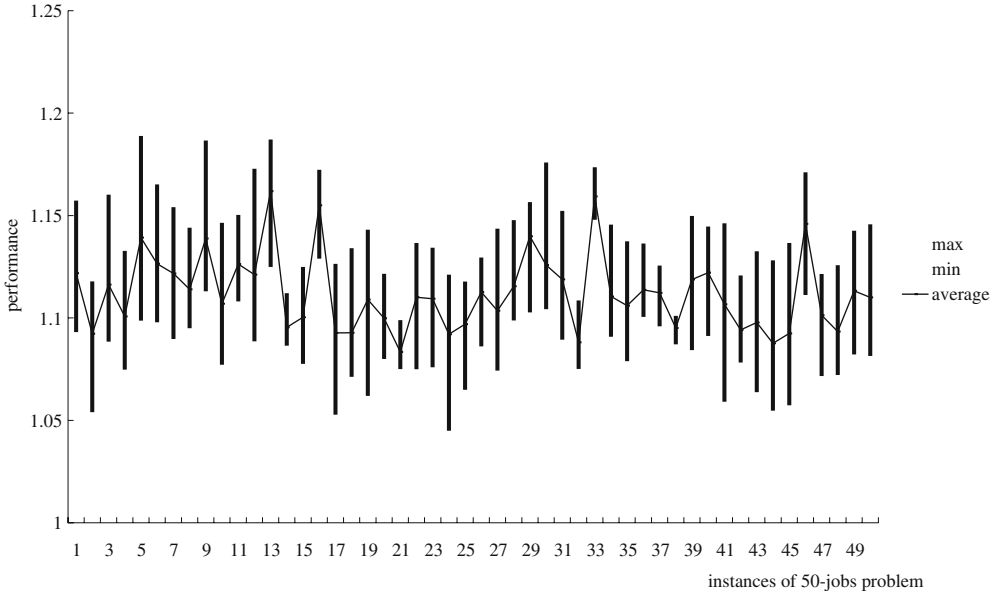

Fig. 8 Performances obtained by A2 for small job sizes

Since the performance of a GA is generally sensitive to the settings of crossover and mutation, a preliminary experiment using the problem of 50 jobs and job sizes with the interval [1, 30] had been conducted to obtain appropriate values. The experimental design is listed in Table 1 and $\frac{sol(GA)}{Lower\ bound}$ of formula is used to measure the performance of GA with difffserent parameter settings where lower bound is obtained by min(r_i)+sol(*FBLPT*) [8]. The preliminary results are shown in Fig. 5. From the figure it is apparent that interaction is present in this experiment and 0.90 of crossover seems to give the best results, regardless of mutation factor. In addition, the performance tends to stabilize when mutation exceeds 10%. The final parameter values chosen for the hybrid GA are listed in Table 2 according to the outcomes of Fig 5.

5 Computational experiments

To compare the algorithms described in the previous sections, these algorithms were tested on problems with 5, 10, 15, 20, 50, and 100 jobs. For each job i, an integer processing time p_i was generated from the uniform distribution [8, 48], an integer release time r_i from the uniform distribution [0, 48], and an integer job size s_i from the uniform distribution [1, 30] and [15, 35]. The maximum capacity of the machine is 40. There were fifty instances for each problem and ten repetitions for each instance in the GA implementation.

Due to the complexity of this problem, which consists of job sequencing, batching, and batch scheduling stages, we use the $H(\alpha, \beta, \gamma)$ notation to represent different approaches. For example, *H(LPT, BFF, ERT)* indicated

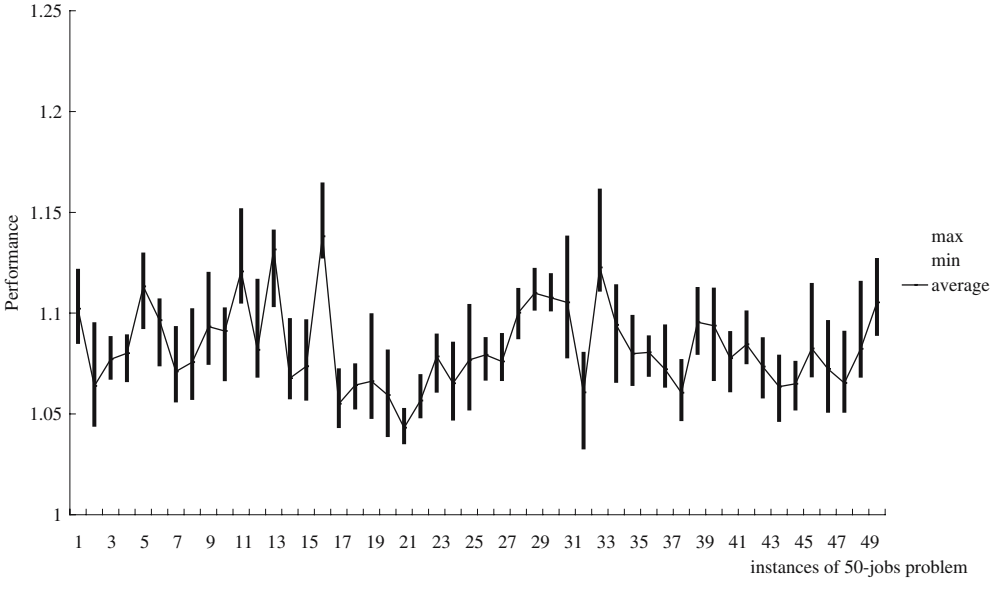

Fig. 9 Performances obtained by A3 for small job sizes

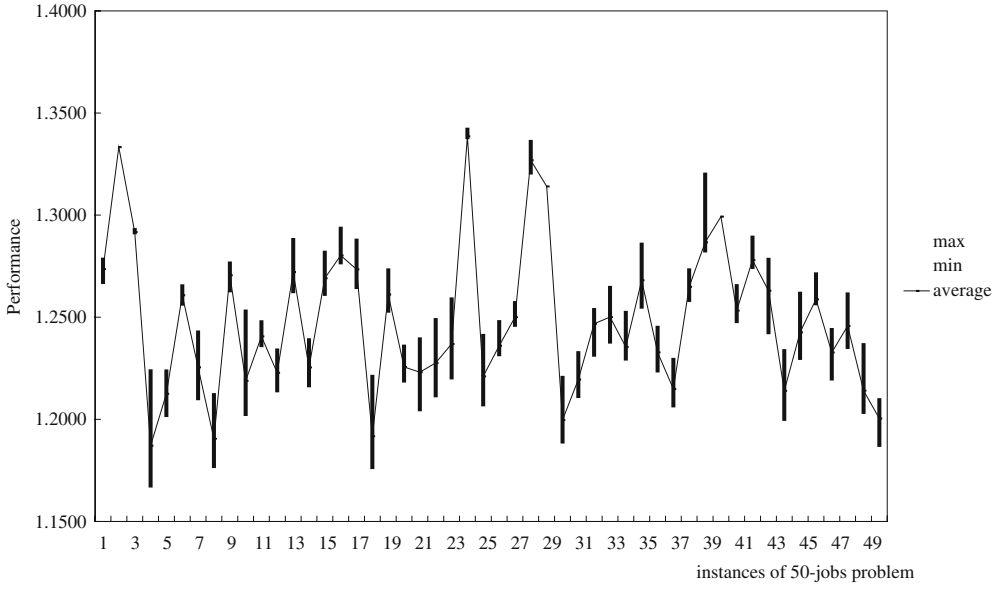

Fig. 10 Performances obtained by A2 for large job sizes

that in stage 1 this approach uses the *LPT* dispatch rule to decide the job sequence, the *BFF* heuristic to form batches in stage 2, and finally assign batches on a single machine by the *ERT* rule in stage 3. Since the results of Uzsoy [10] exhibited that *LPT* dispatch rule on the job sequence decision was better than others, and the *BFF* heuristic is an easy and intuitive method for batching jobs on minimizing makespan criterion, thus in this paper we examined three approached as shown in Table 3. These approaches were coded in C++ language and run on a Pentium *IV* 3.2 GHz computer.

Figures 6 and 7 show the average performance of all the three approaches for $s_i \sim U[1,30]$ and $s_i \sim U[15,35]$, respectively. The performance is computed by dividing the solution of each approach by the lower bound for each instance, i.e., $\frac{sol_i(A_j)}{Lower\ bound_i}$, $i = 1, 2, ..., 50$, $j = 1, 2, 3$. From Figs. 6 and 7, increasing the diversity of job sequence by GA operators is helpful to improve the solution quality and average performance of A2 and A3 more slowly increased than A1 as job number increased. Additionally, the effect of merge-split procedures is significant for small job sizes ($s_i \sim U[1,30]$), especially in large problem cases, but not for big job sizes ($s_i \sim U[15,35]$). This is because bigger job sizes make shifting/swapping jobs less possible, and most batches could accommodate only a few jobs, so that only using GA operators to generate various job sequences would cover the weakness of the *BFF* heuristic. Thus, Fig. 7 shows that the average performance is almost the same for A2 and A3.

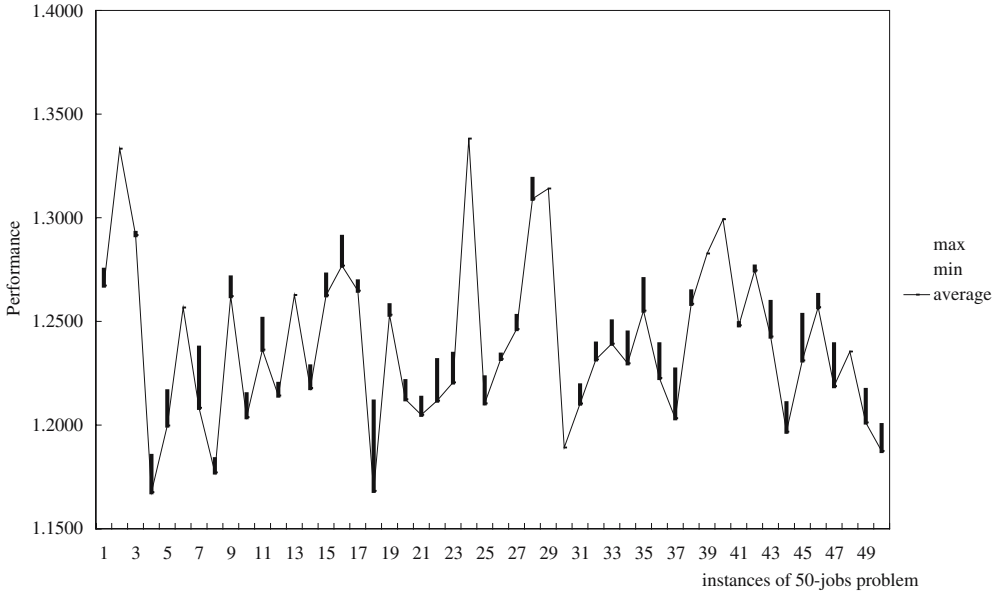

Fig. 11 Performances obtained by A3 for large job sizes

Table 4 Average execution time(s) of the three approaches

n	Job sizes [1,30]			Job sizes [15,35]		
	A1	A2	A3	A1	A2	A3
5	0.0000	0.0295	0.1477	0.0000	0.0285	0.1402
10	0.0000	0.0639	0.2959	0.0000	0.0647	0.3311
15	0.0000	0.1365	0.5769	0.0000	0.1215	0.6029
20	0.0000	0.2981	1.3814	0.0000	0.2228	1.1482
50	0.0000	4.9723	35.7686	0.0000	3.8132	24.2906
100	0.0003	47.8608	453.5126	0.0003	43.6389	420.1930

Figures 8, 9, 10, 11 show the maximum, minimum, and average values obtained over ten repetitions of each instance for the 50-jobs problem by the hybrid GA. From Figs. 8, 9, 10, 11, it is evident that merge-split procedures not only decrease the average value but also decrease the solution variance. Thus we could conclude that hybrid GA incorporating merge-split procedures is worthwhile to implement in real-world batch machine scheduling problems because its robustness.

The average execution times of A1 were less than 1 s for each problem, while A2 and A3 required much more computational time than A1, especially for large problems, as shown in Table 4. This is because there is greater diversity of job sequences as the number of jobs increases, the GA needs more time to converge to the predetermined stopping condition. Additionally, A3 has more execution time than A2 due to in A3 each chromosome needs to proceed merge-split procedures to obtain schedule result for each population (ps=200) of each generation ($maxg$=1000).

6 Conclusions

This paper extends the single batch machine scheduling problem considered by Uzsoy [10] to dynamic job release time. From the results of Uzsoy [10], it was apparent that job sequence has influence on solution quality even for the static problem. This motivated us to develop a hybrid GA which would generate a diversity of job sequences that would decrease the impact and to provide merge-split procedures to implement a local improvement for a given batch schedule. From our computational experiments, the results supported our observation and showed that the hybrid GA obtains satisfactory solution quality. Additionally, merge-split procedures would make the hybrid GA advantage over solution quality improvement and more robust by shifting or swapping jobs between batches.

In this paper we only apply the intuitive batching rule proposed by Uzsoy [10] to form batches, so considering different effective batching rules to group jobs for the problem is of interest for future work. Furthermore, different batch machine problems such as parallel or flow shop environments are also another research direction.

References

1. Ikura Y, Gimple M (1986) Scheduling algorithms for a single batch processing machine. Oper Res Lett 5:61–65
2. Lee C-Y, Uzsoy R, Martin-Vega LA (1992) Efficient algorithms for scheduling semiconductor burn-in operations. Oper Res 40:764–775
3. Li C-L, Lee C-Y (1997) Scheduling with agreeable release times and due dates on a batch processing machine. Eur J Oper Res 96:564–569
4. Sung CS, Kim YH (2003) Minimizing due date related performance measures on two batch processing machines. Eur J Oper Res 147:644–656
5. Chandru V, Lee C-Y, Uzsoy R (1993) Minimizing total completion time on batch processing machines. Int J Prod Res 31:2097–2121
6. Dupont L, Ghazvini FJ (1997) A branch and bound algorithm for minimizing mean flow time on a single batch processing machine. Int J Ind Eng 4:197–203
7. Hochbaum DS, Landy D (1997) Scheduling semiconductor burn-in operations to minimize total flowtime. Oper Res 45: 874–885
8. Lee C-Y, Uzsoy R (1999) Minimizing makespan on a single batch processing machine with dynamic job arrivals. Int J Prod Res 37:219–236
9. Dupont L, Dhaenens-Flipo C (2002) Minimizing the makespan on a batch machine with non-identical job sizes: an exact procedure. Comput Oper Res 29:807–819
10. Uzsoy R (1994) Scheduling a single batch processing machine with non-identical job sizes. Int J Prod Res 32:1615–1635
11. Zhang G, Cai X, Lee C-Y, Wong CK (2001) Minimizing makespan on a single batch processing machine with nonidentical job sizes. Nav Res Log 48:226–240
12. Sung CS, Choung YI (2000) Minimizing makespan on a single burn-in oven in semiconductor manufacturing. Eur J Oper Res 120:559–574
13. Sung CS, Choung YI, Hong JM, Kim YH (2002) Minimizing makespan on a single burn-in oven with job families and dynamic job arrivals. Comput Oper Res 29:995–1007
14. Michalewicz Z (1996) Genetic Algorithms+data structrues = evolution algorithms. Springer, Berlin Heidelberg New York