Int J Adv Manuf Technol (2010) 48:347–354
DOI 10.1007/s00170-009-2281-7

ORIGINAL ARTICLE

Assembly line balancing based on an adaptive genetic algorithm

Jianfeng Yu · Yuehong Yin

Received: 1 February 2009 / Accepted: 19 August 2009 / Published online: 2 September 2009

Abstract An adaptive genetic algorithm is presented as an intelligent algorithm for the assembly line balancing in this paper. The probability of crossover and mutation is dynamically adjusted according to the individual's fitness value. The individuals with higher fitness values are assigned to lower probabilities of genetic operator, and vice versa. Compared with the traditional heuristic algorithms, the adaptive genetic algorithm has effective convergence and efficient computation speed. The computational results demonstrate that the proposed adaptive genetic algorithm is an effective algorithm to deal with the assembly line balancing to obtain a smoother line.

Keywords Assembly line · Balancing · Adaptive probability · Genetic algorithm

1 Introduction

This paper is focused on the assembly line balancing in the reconfigurable assembly systems. Different from the normal manufacturing systems, assembly line balancing (ALB) problem is showing great importance. The topic of assembly line balancing is always attached importance by factory engineers and operation researchers. There are two kinds of ALB problem. If the cycle time is fixed, the objective of ALB problem is to minimize the number of workstations needed (ALB-1 problem), the other is minimizing the cycle time given the number of workstations, task time, and priority sequence (ALB-2 problem). The ALB problem belongs to complex NP-hard class of combinatorial optimization problems. Genetic algorithm has been proven to be very powerful in finding heuristic solutions from a wide variety of applications. Some scientists have studied the assembly line balancing problems using genetic algorithms [1–6]. Chiang studied the assembly line balancing with tabu search algorithm [7]. This paper considers two targets of the balancing problem ALB-1, which are minimizing the number of the workstations and balancing the workload between workstations.

2 Genetic algorithm for solving the combinatorial optimization

2.1 The character of the genetic algorithm

In 1975, John Holland published his book "Adaption in Natural and Artificial Systems", which lays a foundation of genetic algorithms (briefly GAs). Genetic algorithms are an adaptive global optimization probability search methodology, simulating the laws of natural selection and the genetic evolutionary process. The primary character is the population search strategy, information exchanging between the individuals in the population, and the evolution process is independent of the gradient information.

Genetic algorithms keep a group of near-optimal solutions rather than a single-current solution, which are the greatest difference from the other meta-heuristic algorithms. In this sense, genetic algorithms have a natural

J. Yu (✉)
Mechanical and Electrical Products Testing Center,
Jiangsu Entry–Exit Inspection and Quarantine Bureau,
Wuxi 214174, China
e-mail: robotmcu@126.com

Y. Yin
School of Mechanical Engineering,
Shanghai Jiao Tong University,
Shanghai 200240, China

parallelism while simulated annealing algorithm and the tabu search algorithm approach the optimal solution with a single solution. GAs are adaptive methods, which, through many generations, let the natural population evolve by the natural selection according to the principle “the survival of the fittest”. Essentially, during selection process, the individual with higher fitness will have higher tendency to be selected. The quality of the offsprings are improved in the evolution process. GA’s ability comes from technical robustness, so it has been widely applied in the optimization problems.

Different from other traditional optimization algorithms, genetic algorithms deal with a group of individuals which are the potential solutions, the near-optimal solution can be searched in the globe solution space. While all the other methods process a single point in the search space [8]. These kinds of the point-to-point searching algorithms often fall into local optimal solution of single apex in the multi-apex distribution searching space. By contraries the genetic algorithm deals with many individuals of the population at the same time, which is evaluating many solutions in the searching space simultaneously. This feature enables the genetic algorithm to search the space thoroughly and to take less risk of finding a local optimal solution.

2.2 Basic steps of the standard GA

In order to use the genetic algorithm, the following key parts are critical to the success of the GA: gene representation, fitness function design, and probabilities of the genetic operators. The basic steps are as follows:

Step 1: Representation. Generate the initial population randomly. The individual coding could be binary code or real number. These individuals are the gene of chromosome. The size of the population could be arbitrarily chosen according to the complexity of the controlling problem.

Step 2: Fitness evaluation. Decode all the chromosomes and evaluate the objective function of their corresponding candidate solution. Calculate the fitness value of the chromosome according to the given objective function. The fitness represents its probability to survive. The greater the fitness of a chromosome is, the greater the probability to survive.

Step 3: Genetic operations. Generate a new population by the repetition of the following:

Selection: select two agnate chromosomes from the population according to the adaptability. The higher its adaptability is, the more possible the chromosome will be chosen. Usually the gambling wheel selection is used.

Crossover: given the crossover probability, the crossover agnate generates the new offspring. If there’s no crossover, the offspring is the exact copy of the agnate.

Mutation: given the mutation probability, the mutation occurs in a certain place of a chromosome and a new individual is born.

Step 4: Parent selection. Choose pairs of individuals from the population in such a way that those with higher fitness will get more copies. Usually the roulette wheel selection is used.

Step 5: Output. Stopping the calculation cycle and output the optimal solution of current population if satisfying the termination criterion; or return to Step 2.

The nature of GA is the process of dynamic adaptive calculation. Probabilities of genetic operators in standard GA are static during the evolution process, which is against the nature of GA. Adaptive GA is researched, where the dynamic probabilities of crossover and mutation are used [9–12]. Higher crossover probability can exploit bigger solution space, and decrease the possibility of stopping at the non-optimal solution. The mutation probability enhances the proportion of new individuals entering the next generation. If the mutation probability is too low, some useful gene might not be produced, while if too high, too many random genes may cause worse quality of the offsprings compared with their parents. Carefully designing the probabilities of the genetic operators are an important topic in adaptive genetic algorithms.

3 Mathematic model of assembly line balancing

Assembly line balancing problem in this paper is defined as: given a single product, knowing its tasks set and sequence priority constraint relationships, assembly time of each task, and cycle time. The objective is minimizing the workstation numbers and balancing the workload between them. Precedence relationship is always represented by topology network, tasks are expressed by nodes, and direct priority relationships are expressed by arcs.

Cycle time is defined as the maximum time available for each work cycle, which reflects the efficiency of the assembly line. When a fixed cycle time is given, all the workstation must finish its assembly task in the cycle time.

In the reconfigurable assembly system, workstations are can be workers or robots, and accomplishing all the assembly tasks allocated to this workstation. The primary data of assembly system includes: (1) cycle time, (2) priority relationship, and (3) assembly time for each task.

The ideal model is assumed as: (1) each task must be allocated to a workstation respectively, (2) none of the tasks can be allocated twice, (3) operating time on each machine must be less than the cycle time, (4) if task a is prior to task b, task b cannot be allocated before the task a.

Genetic algorithms are well adopted in solving ALB problems [13]. According to the characteristic of ALB problems, four aspects must be considered during the design of GA:

1. The individual representation of GA should be fit for the assembly line balancing.
2. Encoding method should be effective.
3. Simple and practical genetic operators should be implemented to maintain the feasibility of the solutions.
4. Avoid the premature convergence and slow convergence rates.

4 Adaptive genetic algorithm

Genetic operations determine the convergence rates and optimal solution searching. Convergence rate is realized by selection operation, while the optimal solution searching is achieved by crossover and mutation. At the beginning of evolution, because of heavier choosing pressure, the convergence rate is very high. At the anaphase of evolution, because of the small difference between the fitness of each individual, convergence rate becomes lower, and premature convergence may appear. Crossover probability and mutation probability are important factors affecting the performance of GA, which can avoid the premature convergence. The value of parameters in standard GA is unchanged once determined. Adaptive genetic algorithm introduced in this paper uses reconfigurable probabilities of crossover and mutation.

4.1 Encoding

The operation object of GA is gene. The primary job of GA is encoding. Three encoding methods for assembly line balancing are introduced [14], e.g., workstation-oriented representation, sequence-oriented representation, operation-oriented representation:

Task-oriented representation is implemented in this paper. Gene of chromosome is expressed by real number, such as 1, 2, 3,..., n. The length of chromosome, denoted as n, is the total number of tasks to be arranged. Each gene represents one assembly task. When encoding, the diagram of tasks priority constraint is loaded at first, so every gene satisfies the precedence relationship. Each real number between 1 and n must and should appear only one time.

The precedence relationship is represented by a precedence matrix of $n \times n$:

$$p_{ij} = \left[a_{ij}\right]_{n\times n} \tag{1}$$

Where, $a_{ij} = \begin{cases} 1, & \text{task } i \text{ precedes task } j \\ 0, & \text{others} \end{cases}$.

The generation of the initial population must ensure that every individual is a feasible solution, and we will put forward the specific steps to create the initial population according to the precedence constraints.

Step 1: Create initial nodes without precedence relation, we call it as partial order set; at the same time, create a new void sequence set.
Step 2: Choose a task node (for example *ith* node) from partial order set at random, put it into sequence set, and set the *ith* node's row value to zero. Repeat above procedure until partial order set is empty.
Step 3: Judge whether the nodes appeared in the partial order set iterate all the task nodes, if it does, quit the program, if not, go to Step 4.
Step 4: Look for the node without precedence constraint and append it to the partial order set, then return to Step 2.

Note that in Step 3, if all the precedence operations of the successor tasks have already been appeared in the sequence set, the individual definitely is a feasible sequence. If population number is p, then execute the above steps p times.

4.2 Genetic operation

To improve the adaptability of the population, two basic operators, crossover and mutation, are employed to modify the chromosome. The detailed descriptions of the two genetic operators are as follows.

1. Crossover: crossover operation is the leading method in creating new individuals in GA. Traditional two-point crossover or multi-point crossover is not suitable for the combinatorial optimization problems. Three crossover operators (PMX, OX, LOX) are very suited for the combinatorial optimization problems. In the proposed adaptive genetic algorithm, the Partially Mapped Crossover operator is used. It solves the irregularity caused by two-point crossover by repair strategy. It's specific steps are as follows:

   Step 1: Choose two random cut points of parents, and we call the strings defined by these two points as mapped segments
   Step 2: Exchange the two segments of the parents, create a new offspring

Step 3: Definite the mapping relation of the two segments

Step 4: According to the precedence relation, make the offspring feasible through repair strategy

2. Mutation: after crossover, the mutation operator is applied to enhance the genetic diversity in the population and prevents a population prematurely converging at local minima. The mutation of the binary system is not effective in the real number coding, and will cause duplication of the genes. In the assembly line balancing problem, each gene in the chromosome should not be repeated to ensure feasible offsprings. We use a simple and effective mutation method called the feasible insertion which is to replace one or more genes in the chromosome. The procedure may be summarized as follows: we choose two mutation positions in the chromosome randomly. If the gene in the first mutation position does have the sequence constraint relationship with its back neighboring gene, then arrange the two genes in their inverse order, otherwise, the two genes keep their original positions. For the gene in the second mutation position, the same method is used. The concrete steps are described as follows:

Step 1: Choose a chromosome randomly. For example: chromosome 1 3 2 5 4 6.

Step 2: Produce two random number $p1$ and $p2$, which are the mutation positions. If the gene in the first mutation position does have the sequence constraint with its back neighboring gene, then arrange the two genes in their inverse order, otherwise, the two genes keep their original positions. For the gene in the second mutation position, the same method is used. For example: $p1=4$, $p2=2$, moving backward, the new chromosome: 3 1 5 2 4 6.

Step 3: The offspring chromosome replaces the parent chromosome.

4.3 The reconfigurable probability of the crossover and mutation operator

The efficiency and the quality are the contradiction in the design of the genetic algorithm. Their effect is controlled by the probability parameter P_c and P_m. In the standard genetic algorithm, the range of P_c is between 0.5 and 1.0, and P_m often lies between 0.001 and 0.5. The standard genetic algorithm completely neglects the evolution of the fitness where the probability of the crossover and mutation is fixed. In such a circumstance, the excellent individuals in the population will be destroyed by the genetic operations. It brings the negative effects in the efficiency and quality of the algorithm. Crossover operation is run in every generation, in order to improve the convergence rate, we could adjust the probability during the evolution process. That is to say, the higher fitness of the individual, the lower the probability of their crossover and mutation, the adaptive probability can ensure that the algorithm can obtain the optimal solution. The probability of the genetic operators is adapted by itself, when the fitness of the individual is increased, the probability of the crossover and mutation should be reduced to speed up the convergence; and vice verse [15–17]. In this paper we give a reconfigurable strategy for the probability of the crossover and mutation. The probabilities of the crossover and mutation are outlined below:

$$P_c = \begin{cases} k_1 + (k_3 - k_1)*(f_{\max} - f_c)/(f_{\max} - f), & f_c \geq \overline{f} \\ k_3, & f_c < \overline{f} \end{cases}. \tag{2}$$

$$P_m = \begin{cases} k_2 + (k_4 - k_2)*(f_{\max} - f_i)/\left(f_{\max} - \overline{f}\right), & f_i \geq \overline{f} \\ k_4, & f_i < \overline{f} \end{cases}. \tag{3}$$

In Eqs. 2 and 3, k_1, k_2, k_3, and k_4 are all less than 1 to restrain P_c, P_m between 0.0 and 1.0. f_c is the bigger fitness in two individuals of the crossover. f_i is the fitness of the ith individual in the population. $f_{\max}$ is the maximum fitness of the current population, $\overline{f}$ is the average fitness of the current population.

When GA converges, the difference of the fitness between the individuals in each population will be smaller and smaller. The sum of the fitness difference between the ith individual and other individuals is expressed as Eq. 4:

$$C(f_i) = \sum_{j=1}^{n} |f_i - f_j|, i \neq j. \tag{4}$$

The standard fitness difference is equal to the sum of fitness difference divided by $(n-1)\max_j|f_i - f_j|$ [10], which is described as Eq. 5:

$$\widetilde{C}(f_i) = \frac{\sum_{j=1}^{n} |f_i - f_j|}{(n-1)\max_j|f_i - f_j|}, \quad i \neq j. \tag{5}$$

Thus $\widetilde{C}(f_i)$ is between 0 and 1. It is effective to adjust the mutation probability by $\widetilde{C}(f_i)$, when $\widetilde{C}(f_i)$ increased, it shows that the differences between this individual and the other individuals in the current population are enlarged, we should use the higher mutation probability to mutate this chromosome. According to the above description, we could set the mutation probability of the individual according to Eq. 6:

$$P_m = \begin{cases} k_2 + (k_4 - k_2) * (f_{\max} - f_i) * \widetilde{C}(f_i) / \left(f_{\max} - \overline{f}\right), f_i \geq \overline{f} \\ k_4, \qquad f_i < \overline{f} \end{cases} . \tag{6}$$

Through well-designed parameters according to the Eqs. 2 and 6, we protect the solution which has high fitness, while the individuals under the average fitness should be genetic operated with higher probabilities. The adaptive probabilities strategy pushes the population to approach to the optimal solution. Therefore the adaptive probabilities strategy can prevent premature convergence and speed up the GA convergence rate. To those individuals far from the optimal solution, we use the higher value of P_c and P_m to exploit the searching space; to those individuals close to the optimal solution, we use the lower value of P_c and P_m to ensure the algorithm not to converge into the local optimal solution.

4.4 Multi-population selection operator

We also call selection operator as “reproduction”. Its main purpose is to produce the new offsprings from the current population following the principle of “the survival of the fittest”. So the selection operation should ensure the individual with high fitness to be chosen, and maintain the diversity of population in order to prevent the premature convergence. We enlarge population number by two times in the selection process. At the same time, using elite preservation strategy, we search the individual with lowest fitness in the new population, and replace it by the one with highest fitness in the parent population. The best individual in the population always leads the direction of the evolution. The elite preservation strategy speeds up the convergence rate.

4.5 Fitness

The fitness is the criterion to represent the ability of the individual, and it is the foundation of “survival of fittest” in the genetic algorithm. Therefore, it is the driving power for the genetic algorithm. In general, the fitness function of the individual is the objective function. Sometimes, the fitness function is the transformation of the objective function.

The value of the fitness is usually a positive number. The greater the fitness of a individual, the higher possibility to survive. So we should transfer the fitness function into a maximum problem when dealing with the minimum problem. Various objective criteria have also been presented to measure the quality of a scheduling in ALB problems [18]. In this paper, the objective functions of the assembly line balancing should be optimized, one is to minimize the number of the workstations, the other is to balance the workload between the workstations. However, two different solutions at the same workstation number may have different balancing result. For example, one assembly line has three workstations, and the assembly time for each workstation is 30–40–50 or 50–20–50. We regard the solution 30–40–50 is better than 50–50–20, because the former solution is more balancing than the latter. According to this, we consider two objectives, the minimized workstation number and the balanced workstation load. Thus the fitness function f can be defined as Eq. 7:

$$f = S_{\max} - \sqrt{\sum_{k=1}^{n} (C_{\max} - T_k)^2} - 2n \tag{7}$$

n: the number of workstation, $C_{\max}$: the desired cycle time, T_k: the assembly time of the kth task, $S_{\max}$: a big constant, to ensure that the objective function is always non-negative number.

The first part $\sqrt{\sum_{k=1}^{n} (C_{\max} - T_k)^2}$ in Eq. 7 is to find the best balancing solution at the same number of workstation situation, and we call it as the smoothness index. When it equals to zero, it denotes that the perfect balancing is achieved. The second part $2n$ is to minimize the number of workstations. Here, we consider it more important than the first. So the weight factor is 2. The higher the fitness function, the smaller the number of workstations, also the more balanced workload between the workstations.

5 Numerical illustrations

Example 1: Supposed the product’s working procedure as follows. The productivity of assembly line is 53.33 U/h, the efficiency is 96%, so the cycle time is 1.08 min, and the task sequence relationship is illustrated in Fig. 1. Here, we use the Kilbridge–Wester algorithm [19] and adaptive GA to allocate the tasks to the workstations, considering the best workload balancing and the minimized workstation number.

The results computed by the heuristic Kilbridge–Wester algorithm is shown in the Table 1, the results computed by adaptive GA is shown in Table 2. We can see that all the tasks allocation is satisfied with t the sequence relationship shown in Fig. 1.

Compared the Table 1 with the Table 2, we can find that the two algorithms output the same number of workstation number, both are five workstations. The tasks combination allocated by AGA is shown in Fig. 2. The smoothness

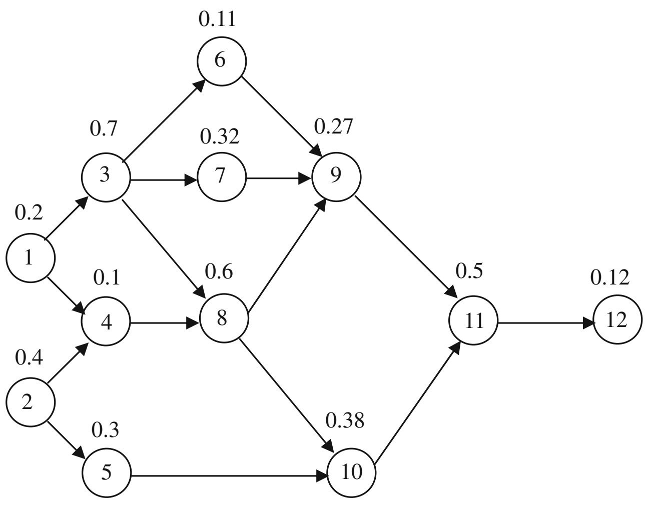

Fig. 1 Sequence precedence diagram for 12 tasks

Table 2 Stations assigned according to the AGA

Workstation no.	Task node	Assembly time (min)	Total assembly time (min)
1	1	0.2	0.7
	2	0.4	
	4	0.1	
2	3	0.7	0.81
	6	0.11	
3	7	0.32	0.62
	5	0.3	
4	8	0.6	0.98
	10	0.38	
5	9	0.27	0.89
	11	0.5	
	12	0.12	

index from the balancing results of the adaptive GA is $\sqrt{\sum_{k=1}^{n}(S_{\max}-S_k)^2}=0.245$, while the smoothness index from the balancing results of the Kilbridge–Wester algorithm is $\sqrt{\sum_{k=1}^{n}(S_{\max}-S_k)^2}=0.3094$. That is to say, the adaptive GA can produce better balancing effect than the Kilbridge–Wester algorithm.

Example 2: Supposed the 25 tasks precedence relationship is shown in Fig. 3. The cycle time for the workstation is 30 min. Here, we use the Monte Carlo simulation algorithm [20] and adaptive GA to allocate the tasks to the workstations, considering the best workload balancing and the minimized workstation number.

The results computed by the Monte Carlo simulation method is shown in Fig. 4, the results computed by AGA is shown in Fig. 5. All the 25 tasks are both allocated on the six workstations.

The smoothness index from the balancing results of the AGA is $\sqrt{\sum_{k=1}^{n}(S_{\max}-S_k)^2}=3.316$, The smoothness index from the balancing results of Monte Carlo algorithm is $\sqrt{\sum_{k=1}^{n}(S_{\max}-S_k)^2}=4.242$. The results show that the genetic algorithm reduces the smoothness index and thereby results in a smoother line than Monte Carlo algorithm.

6 Conclusions

This paper puts forward an adaptive genetic algorithm, carries out further research on the adaptive crossover

Table 1 Stations assigned according to the Kilbridge–Wester method

Workstation no.	Task node	Assembly time (min)	Total assembly time (min)
1	2	0.4	1.0
	1	0.2	
	5	0.3	
	4	0.1	
2	3	0.7	0.81
	6	0.11	
3	8	0.6	0.92
	7	0.32	
4	10	0.38	0.65
	9	0.27	
5	11	0.5	0.62
	12	0.12	

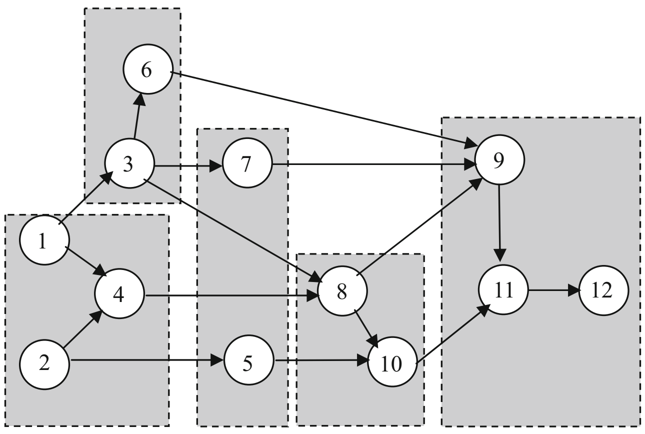

Fig. 2 Workstation allocation diagram

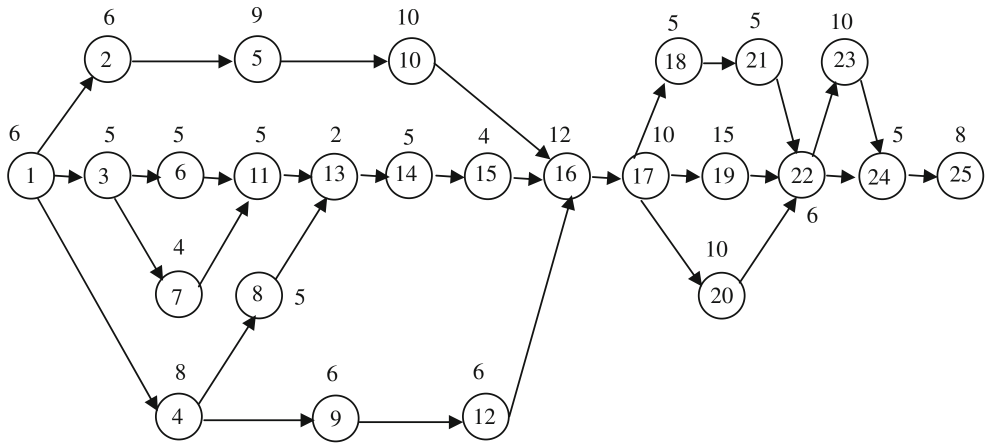
Fig. 3 Sequence precedence diagram for 25-task case

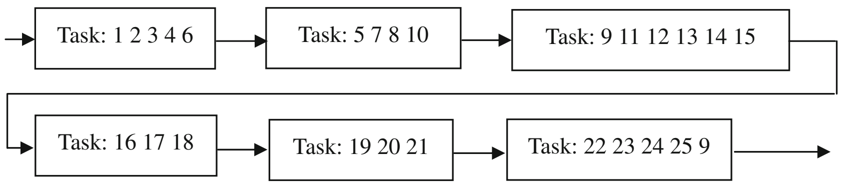

Fig. 4 Tasks assigned according to the Monte Carlo simulation method

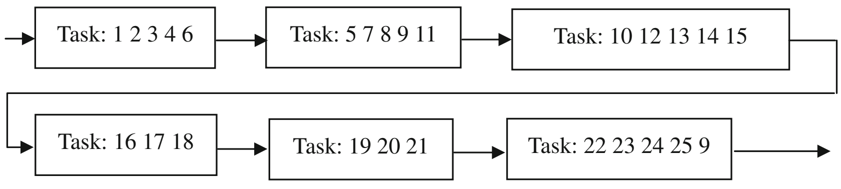

Fig. 5 Tasks assigned according to the AGA

probability and mutation probability. This paper provides the sequence-oriented coding solution to ensure that chromosome is subject to the task precedence. Two computation examples demonstrate that adaptive genetic algorithm is better than the heuristic Kilbridge–Wester algorithm and Monte Carlo algorithm in solving the ALB-1 problem. The adaptive genetic algorithm provides an effective and practical method to solve the ALB-1 problem, which can results in a smoother assembly line.

Acknowledgement This research work was partially supported by the National Natural Science Fund of China (Grant No. 60205006). The authors are grateful to Dr. Song Yin, Jiareng Song, Shiyi Chen, and Tianpu Han for their help in preparing this paper.

References

1. Kim YK, Kim YJ, Kim Y (1996) Genetic algorithms for assembly line balancing with various objectives. Comput Ind Eng 30 (3):397–409
2. Sabuncuoglu I, Erel E, Tanyer M (2000) Assembly line balancing using genetic algorithms. J Intell Manuf 11:295–310
3. Ubinovitz J, Levitin G (1995) Genetic algorithm for assembly line balancing. Int J Prod Econ 41:343–354
4. Gonçalves JF, Almeida JRD (2002) A hybrid genetic algorithm for assembly line balancing. J Heuristics 8:629–642
5. Ponnambalam SG, Aravindan P, Naidu GM (2000) A multi-objective genetic algorithm for solving assembly line balancing problem. Int J Adv Manuf Tech 16:341–352
6. Aytug H, Khouja M, Vergara FE (2003) Use of genetic algorithms to solve production and operations management problems: a review. Int J Prod Res 41(17):3955–4009
7. Chiang WC (1998) The application of a tabu search metaheuristic to the assembly line balancing problem. Ann Oper Res 77:209–227
8. Khoo LP, Loi MY (2002) A tabu-enhanced genetic algorithm approach to agile manufacturing. Int J Adv Manuf Technol 20:692–700
9. Oyama A, Obayashi S, Nakamura T (2001) Real-coded adaptive range genetic algorithm applied to transonic wing optimization. Appl Soft Computing 1(3):179–187
10. Wu QH, Cao YJ, Wen JY (1998) Optimal reactive power dispatch using an adaptive genetic algorithm. Electrical Power & Energy Systems 20(8):563–569
11. Chew EP, Ong CJ, Lim KH (2002) Variable period adaptive genetic algorithm. Comput Ind Eng 42:353–360
12. Mak KL, Wong YS, Wang XX (2000) An adaptive genetic algorithm for manufacturing cell formation. Int J Adv Manuf Technol 16:491–497
13. Wong WK, Mok PY, Leung SYS (2006) Developing a genetic optimization approach to balance an apparel assembly line. Int J Adv Manuf Technol 28:387–394
14. Guo ZX, Wong WK, Leung SYS, Fan JT, Chan SF (2008) A Genetic-algorithm-based optimization model for solving the flexible assembly line balancing problem with work sharing and workstation revisiting. IEEE Trans Syst Man Cybern C Appl Rev 38(2):218–220
15. Srinivas M, Patnaik LM (1994) Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Trans Syst Man Cybern C Appl Rev 24(4):656–667
16. Raman S, Patnaik LM (1996) Performance-driven MCM partitioning through an adaptive genetic algorithm. IEEE Trans Very Large Scale Integr (VLSI) Syst 4(4):434–444
17. Tang HC (2003) Using an adaptive genetic algorithm with reversals to find good second-order multiple recursive random number generators. Math Methods Oper Res 57:41–48
18. Guo ZX, Wong WK, Leung SYS, Fan JT, Chan SF (2008) A genetic-algorithm-based optimization model for scheduling flexible assembly lines. Int J Adv Manuf Technol 36:156–168
19. Ponnambalam SG, Aravindan P, Naidu GM (1999) A comparative evaluation of assembly line balancing heuristics. Int J Adv Manuf Tech 15:577–586
20. Nkasu MM, Leung KH (1995) Computer-integrated manufacturing assembly system design. Integr Manuf Syst 6(6):4–14