Open access • Journal Article • DOI:10.1023/A:1008923410076

Assembly line balancing using genetic algorithms — Source link

Ihsan Sabuncuoglu, Erdal Erel, M. Tanyer

Institutions: Bilkent University

Published on: 01 Jun 2000 - Journal of Intelligent Manufacturing (Kluwer Academic Publishers)

Topics: Heuristics, Heuristic, Genetic algorithm, Simulated annealing and Chromosome (genetic algorithm)

Journal of Intelligent Manufacturing (2000) **11**, 295–310

Assembly line balancing using genetic algorithms

I. SABUNCUOGLU,† E. EREL* and M. TANYER†

†*Department of Industrial Engineering and *Department of Management, Bilkent University, Bilkent, Ankara 06533, Turkey*

Received February 22, 1998 and accepted December 20, 1998

Assembly Line Balancing (ALB) is one of the important problems of production/operations management area. As small improvements in the performance of the system can lead to significant monetary consequences, it is of utmost importance to develop practical solution procedures that yield high-quality design decisions with minimal computational requirements. Due to the NP-hard nature of the ALB problem, heuristics are generally used to solve real life problems. In this paper, we propose an efficient heuristic to solve the deterministic and single-model ALB problem. The proposed heuristic is a Genetic Algorithm (GA) with a special chromosome structure that is partitioned dynamically through the evolution process. Elitism is also implemented in the model by using some concepts of Simulated Annealing (SA). In this context, the proposed approach can be viewed as a unified framework which combines several new concepts of AI in the algorithmic design. Our computational experiments with the proposed algorithm indicate that it outperforms the existing heuristics on several test problems.

Keywords: Assembly systems, assembly line balancing, artificial intelligence, genetic algorithms, simulated annealing

1. Introduction

An assembly line consists of a sequence of stations performing a specified set of tasks repeatedly on consecutive product units moving along the line at constant speed. Each unit spends the same amount of time, called the cycle time in every station, the production rate is the reciprocal of this cycle time. Tasks or operations are indivisible elements of work which have to be performed by consuming a fixed amount of time to assemble a product. Due to technological restrictions, precedence constraints specifying the sequence of tasks have to be considered. These constraints are represented by a precedence graph consisting of nodes for the tasks and arcs for the precedence relations. The Assembly Line Balancing (ALB) problem is to determine the allocation of the tasks to an ordered sequence of stations such that each task is assigned to exactly one station, no precedence constraint is violated, and some selected performance measure is optimized (e.g., minimize the number of stations).

Since the ALB problem falls into the NP-hard class of combinatorial optimization problems, numerous research efforts have been directed towards the development of computer efficient approximation algorithms or heuristics (Ghosh and Gagnon, 1989). The common characteristic of all the heuristic search methodologies is the use of problem-specific knowledge intelligently to reduce the search efforts. In this context, GAs are intelligent random search mechanisms that are applied to various combinatorial optimization problems such as scheduling, TSP, and ALB. The existing studies in the literature have indicated that GA can be used as a very effective search technique in solving difficult problems because of its ability to move from one solution set to another and flexibility to incorporate the problem specific characteristics. To achieve these benefits, standard GA operators should be properly modified and adapted to the problem domain. In this paper we propose such a new GA structure and related operators to solve the ALB problem. In fact, our test results on the benchmark problems show that the proposed GA approach yields better ALB schedules than the existing GA methods and other traditional heuristics. Furthermore, the computation time of the proposed method is reasonably low (less than 2 seconds for about 50 tasks ALB problems) that GA can be effectively used in solving real size problems.

The rest of the paper is organized as follows. First, we review the relevant literature. Then we propose a new GA-based algorithm. We also integrate GA and simulated annealing (SA), working together to achieve a better search. Next, we measure the performance of our algorithm on a number of test problems and compare it with the heuristics reported to perform well. Finally, we summarize the important findings and outline the further research directions.

2. Relevant literature

In this section, we first review the traditional studies in the literature and then discuss GA-based approaches.

2.1. *ALB literature*

Since the ALB problem was first formulated by Helgeson *et al.* (1954), many solution approaches have been proposed. Ghosh and Gagnon (1989) classify these studies into four categories: Single Model Deterministic (SMD), Single Model Stochastic (SMS), Multi/Mixed Model Deterministic (MMD), and Multi/Mixed Model Stochastic (MMS). In this paper, we consider the SMD category, which assumes dedicated, single-model assembly lines where the task times are known deterministically and an efficiency criterion is optimized. SMD has been the most researched category, as evidenced by a large number of articles published in the literature (64 articles since 1983) (Ghosh and Gagnon, 1989). A summary of this research work in this category is as follows.

There have been a number of attempts to optimally solve the SMD version of the problem using linear programming (LP) (Salveson, 1955), integer programming (IP) (Bowman, 1960; Klein, 1963; Patterson and Albracht, 1975; Talbot and Patterson, 1984), dynamic programming (DP) (Jackson, 1956; Held *et al.*, 1963; Schrage and Baker, 1978), and branch-and-bound (B&B) approaches (Jackson, 1956; Johnson, 1981; Wee and Magazine 1981). Since the optimal solution of even a modest size problem (e.g., with 100 tasks) is impossible by the exact methods, a considerable research effort has been spent to develop heuristic approaches. Among them, most notable ones are: Dar-El's MALB (1973), Dar-El and Rubinovitch's MUST (1979), Baybars' LBHA (1986), Tonge's (1965), Moodie and Young's (1965), and Nevins' (1972) heuristics. Baybars (1986) compares his heuristic with Tonge's (1965), Moodie and Young's (1965), and Nevins' (1972) heuristics on Tonge's problems. We will use the same problem set to measure the performance of the proposed algorithm.

2.2. *GA approaches to the ALB problem*

GAs are adaptive methods which can be used to solve optimization problems. They are based on genetic processes of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection and survival of the fittest. In nature, individuals with the highest survival rate have relatively a large number of offsprings; that is, the genes from the highly adapted or fit individuals spread to an increasing number of individuals in each successive generation. The strong characteristics from different ancestors can sometimes produce super-fit offspring, whose fitness is greater than that of either parent. In this way, species evolve to become more and more well-suited to their environment. Holland (1975) showed that a computer simulation of this process of natural adaptation could be employed for solving optimization problems. Goldberg (1989) presented a number of applications of GAs to search, optimization and machine learning problems.

In general, the power of GAs comes from the fact that the technique is robust, and can deal with a wide range of problem areas. Although GAs are not guaranteed to find the optimal solution, they generally find good solutions with reasonable computational requirements.

To the best of our knowledge, there are only three published papers in literature which solve ALB problem using GA; two of them work on the deterministic (SMD) problem and the other works on the stochastic problem (SMS). We now present a review of these articles in chronological order.

The first attempt was made by Leu *et al.* (1994). In this study, the authors use solutions of heuristic procedures in the initial population. They also demonstrate the possibility of balancing assembly lines with multiple criteria and side constraints such as, allocating a task in a station by itself. According to the authors, the GA approach has two advantages: (i) GAs search a population rather than a single point and this increases the odds that the algorithm will not be trapped in a local optimum since many solutions are considered concurrently, and (ii) GA fitness functions may take any form (i.e., unlike gradient methods that have differentiable evaluation functions) and several fitness functions can be utilized simultaneously.

In the second study, Anderson and Ferris (1994) showed the effective use of GAs in the solution of combinatorial optimization problems, working specifically on the ALB problem. The authors first describe a fairly standard implementation for the ALB problem. Then an alternative parallel version of the algorithm for use on a message passing system is introduced. Their aim is not to demonstrate the superiority of a GA over the traditional methods, but rather to give some indications for the potential use of this technique in combinatorial optimization problems. Thus, the authors do not compare the GA with well known heuristics, but only with a neighborhood search scheme with multiple restarts in which the GA is found to be better than this method.

Suresh *et al.* (1996) used a GA to solve the SMS version of the ALB problem. The ability of GAs to consider a variety of objective functions is regarded as the major feature of GAs. A modified GA working with two populations, one of which allows infeasible solutions, and exchange of specimens at regular intervals is proposed for handling irregular search spaces, i.e., the infeasibility problem due to precedence relations. The authors claim that a population of feasible solutions would lead to a fragmented search space, thus increasing probability of getting trapped in a local minima. They also state that infeasible solutions can be allowed in the population only if genetic operators can lead to feasible solutions from an infeasible population. Since a purely infeasible population may not lead to a feasible solution in this particular problem, two alternative populations, one purely feasible and one allowing a fixed percentage of infeasible chromosomes, are combined in a controlled pool to facilitate the advantages of both of them. Certain chromosomes are exchanged at regular intervals between the two populations, the exchanged chromosomes have the same rank of fitness value in their own populations. The results of the experiments indicate that the GA working with two populations gives better results than the GA with one feasible population.

3. The proposed GA-based approach

The three studies summarized in the previous section demonstrate that GA is a promising intelligent heuristic for the ALB problem. In this study, we direct our research effort towards exploiting the characteristics of the ALB problem to further improve the existing GA structures. After presenting the initial GA structure, we explain the proposed GA approach in detail.

3.1. *The characteristics of the initial GA*

The structure of our GA is similar to Whitley and Kauth's (1988) GENITOR, as it performs only one crossover operation at each iteration. The initial structure (i.e., without dynamic partitioning and SA-controlled elitism) of our algorithm is as follows.

```
Initial Genetic Algorithm
  Generate initial population
  repeat
    Choose two parents for recombination
    Apply mutation with $R_m$ probability or
    crossover with $1 - R_m$ probability
    Replace parents with offsprings
  until Stopping_condition is reached
  Take the best-fit chromosome of the final
  population as the solution
```

Some of the characteristics of the proposed GA are devised with the inspiration taken from current examples in the literature. We describe these characteristics as follows.

(1) *Coding:* Each task is represented by a number that is placed on a string (i.e., chromosome) with the string size equal to the number of tasks. The tasks are ordered on the chromosome relative to their order of processing. Then the tasks are allocated into stations such that the sum of the task times in each station does not exceed the cycle time. This coding scheme is demonstrated in Fig. 1 for a 7-task problem.

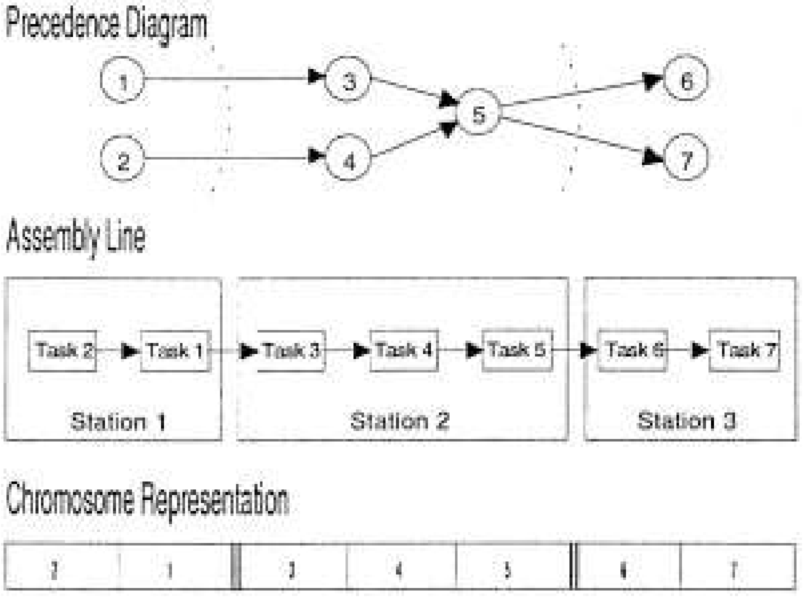


Fig. 1. Coding the chromosome representation of an assembly line.

(2) *Fitness function:* The objective of the ALB problem considered in this paper is to minimize the number of stations, however, given two different solutions with the same number of stations, one may be ''better balanced'' than the other. For example, a line with three stations may have stations times as 30-50-40 or 50-50-20. We consider the 30-50-40 solution to be superior (better balanced) to the 50-50-20 solution. Hence, we used a fitness function that consists of two objectives, i.e., minimizing the number of stations and obtaining balanced station:

$$Fitness\ Function = 2\sqrt{\frac{\sum_{k=1}^{n}(S_{\max} - S_k)^2}{n}} + \frac{\sum_{k=1}^{n}(S_{\max} - S_k)}{n} \quad (1)$$

where n is the number of stations, $S_{\max}$ is the maximum station time, and S_k is the kth station time. The first part of the fitness function aims to find the best balance among the solutions that have the same number of stations while the second part minimizes the number of stations in the solution. Since we arbitrarily assume that the first objective is more critical than the second, we multiply it by two.

(3) *Initial Population:* The initial population is generated randomly by assuring feasibility of precedence relations.

(4) *Crossover and Mutation:* Whether to perform crossover or mutation depends on a certain probability, i.e., if the probability of recombining is 98% then the probability of mutating is 2%. The crossover (recombination) operator is a variant of Davis' (1985) *order crossover* operator. The two parents that are selected for crossover are cut at two random cut-points. The offspring takes the same genes outside the cut-points at the same location as its parent and the genes in between the cut-points are scrambled according to the order that they have in the other parent. This procedure is demonstrated in the example (Fig. 2). The major reason that makes this crossover operator very suitable for ALB is that it assures feasibility of the offspring. Since both parents are feasible, both children must also be feasible. Keeping a feasible population is a key to ALB problem since preserving feasibility drastically reduces computational effort.

The mutation operator of Leu *et al.* (1994) is *scramble mutation*, that is, a random cut-point is selected and the genes after the cut-point are randomly replaced (scrambled), assuring feasibility. Elitism, i.e., replacing a parent with an offspring only

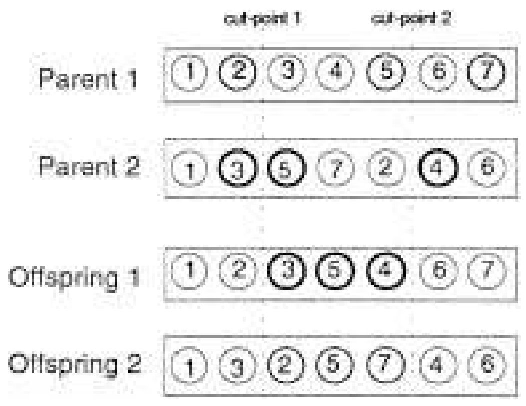


Fig. 2. Our crossover operator.

if the offspring is better than the parent, is applied to both the crossover and the mutation procedures. Both of these operators are the same as Leu *et al.*'s (1994) crossover and mutation operators.

(5) *Scaling:* The fitness scores need to be scaled such that the total of the scaled fitness scores are equal to 1, in order to activate the selection procedure (i.e., roulette wheel selection). Since our objective is to minimize the fitness scores, we need to assign the highest scaled fitness score to the lowest fitness score and vice versa, to assign a probability of selection that is proportional to the fitness of chromosomes. We achieve this by subtracting each fitness value from the double of the highest (worst) fitness value in the population and assigning the subtrahend as the new fitness value of that chromosome. Then, by dividing each new fitness score by the total of new fitness scores, we scale the fitness scores such that their total equals to 1.

(6) *Selection Procedure:* We use a well known procedure called ''roulette wheel selection''. Each chromosome consisting of an interval proportional to its scaled fitness score are placed next to each other on the [0,1] interval. Then, a uniform random number in the [0,1] interval is generated, and the chromosome which is assigned to the interval corresponding to the random number is selected. This procedure selects chromosomes proportional to their fitness scores.

(7) *Stopping Condition:* The algorithm terminates after a certain number of iterations. We used 500, 1000, and 2000 values for the number of iterations parameter.

3.2. *The proposed dynamic partitioning technique*

In this section, we develop a new method called Dynamic PArtitioning (DPA) that modifies chromosome structures of GAs to save CPU time. DPA modifies the chromosome structure by allocating tasks to stations (i.e., freezing certain tasks) that satisfy some criteria, and continues with the remaining unfrozen tasks. Consequently, DPA allows the GA to focus on the remaining tasks during the search and saves a considerable amount of computation time. In what follows, we use ''without DPA'' to refer to the traditional GA discussed in Section 3.1 and ''with DPA'' to refer to the GA with dynamic partitioning.

3.2.1. *Motivation*

Although a typical GA developed for the ALB problem can be a fast problem solver (e.g., the proposed GA solves a 50 task problem after 500 iterations in approximately 1.5 seconds on a Pentium 133 PC), it needs a careful experimental design to tune the parameters for each type of the ALB problem. Hence, it has to be run a number of times (in the order of ten thousands) that can consume significant CPU times. The main motivation behind the development of the DPA methodology is to reduce the CPU times in spite of the expected deterioration in the performance, i.e., the final fitness score. Surprisingly, as it will be discussed in the next section, the performance is improved as well. This indicates that DPA causes the GA to work out more effectively with the remaining ''a fewer number of tasks'' after each freezing or partitioning.

3.2.2. *Implementation*

To preserve the precedence relations between the remaining tasks, we consider freezing at the first and the last stations (i.e., the genes at the beginning and at the end of the chromosome are considered as potentially freezable). The second criteria for freezing is to achieve an optimal station time at the potentially freezable stations. This optimality condition depends on the fitness function. The freezing criteria that best fits to our fitness function is:

$$\begin{aligned} \frac{|S^* - S_i|}{S^*} < DPC, \quad i = 1, n; \\ DPC = 0.01, 0.02, 0.03, \ldots \end{aligned} \tag{2}$$

where

$$S^* = \frac{\sum_{i=1}^{n} S_i}{n^*}$$

n^* is the minimum possible number of stations, i.e.,

$$n^* = \left\lceil \frac{\sum_{i=1}^{n} S_i}{CT} \right\rceil$$

and CT is the cycle time.

The DPC (*Dynamic-Partitioning-Constant*) parameter enables us to fine-tune our algorithm (i.e., it adjusts the accuracy of the station freezing criteria). When it increases, the average number of partitioning per run also increases, resulting in computation time savings, but we may end up with a poorer solution (i.e., worse final fitness scores).

As described above, the two criteria for DPA are checked at the end of each iteration. If the first or the last station satisfies the criteria, then this (these) station(s) is (are) frozen and the GA goes on to the next iteration with the unfrozen tasks only. Since the length of the chromosome decreases after each freezing (or partitioning), the GA program spends less time per iteration in the remaining iterations.

The population size, i.e., the number of chromosomes in the GA population, stays fixed throughout the procedure. The best-fit chromosome, yielding the best solution, is checked for the DPA criteria at each iteration. If it satisfies the criteria, DPA is applied to the best-fit chromosome and the frozen genes (tasks) are deduced from all the other chromosomes of the population. This does not create any infeasibility for the precedence constraints since the frozen tasks are either at the beginning or at the end of the partitioned chromosome.

The DPA mechanism is explained with an example depicted in Fig. 3. In this example, DPA criteria are satisfied for both the first and the last stations at the 45th iteration. Hence, tasks 1, 2, 13, 15, and 16 are frozen. Then, the GA balances the remaining eleven tasks, ignoring the frozen five. At the 136th iteration, only the first station satisfies the DPA criteria, and hence the tasks belonging to this station (i.e., tasks 7, 11) are frozen. These frozen tasks are then added on to the best-fit chromosome of the final iteration in the order that they were frozen.

It is presumed that if DPA is applied starting with the first iteration, then we might do early freezing which would bind us to a local optima. In order to prevent this, we use a warm-up period that allows the initial random population to achieve a considerable fitness score prior to partitioning.

4. Experimental conditions

To investigate the effectiveness of DPA, we solve 30 different ALB problems that are generated in a similar way as discussed in Leu *et al.* (1994). In addition, we

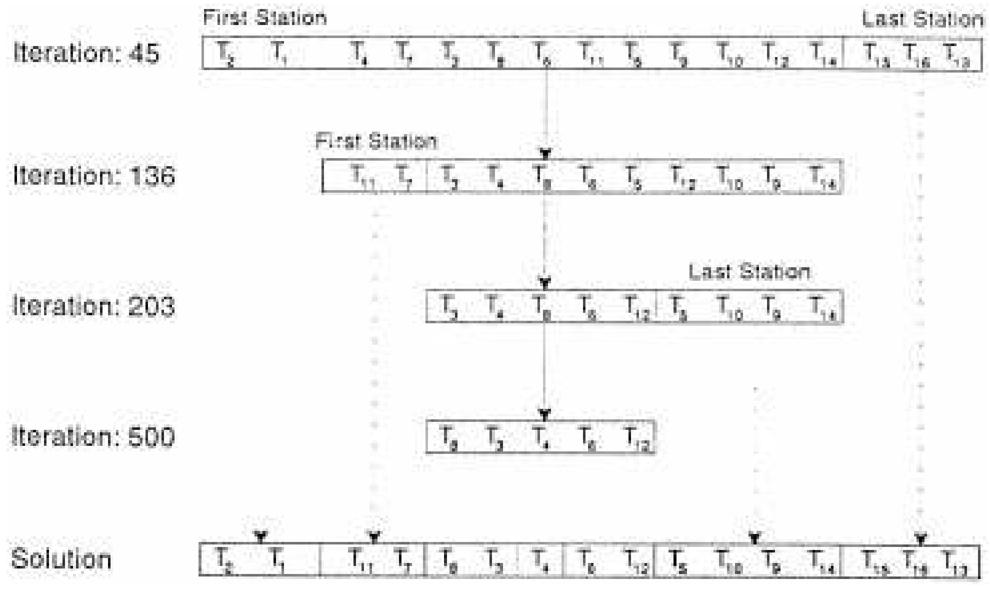

First Station
Last Station
Iteration: 45
T_2 T_1 T_4 T_7 T_3 T_8 T_6 T_{11} T_5 T_9 T_{10} T_{12} T_{14} T_{15} T_{16} T_{13}
First Station
Iteration: 136
T_{11} T_7 T_3 T_4 T_8 T_6 T_5 T_{12} T_{10} T_9 T_{14}
Last Station
Iteration: 203
T_3 T_4 T_8 T_6 T_{12} T_5 T_{10} T_9 T_{14}
Iteration: 500
T_8 T_3 T_4 T_6 T_{12}
Solution
T_2 T_1 T_{11} T_7 T_8 T_3 T_4 T_6 T_{12} T_5 T_{10} T_9 T_{14} T_{15} T_{16} T_{13}

Fig. 3. Illustration of dynamic partitioning.

measure the effects of different DPA and GA parameters on the solutions.

Thirty problems each consisting of 50 tasks are randomly generated for three Flexibility-Ratios (F-Ratios) of 10%, 50%, and 90%. F-Ratio is a measure of the precedence relations among the tasks and calculated as follows

$$F - \text{Ratio} = \frac{2 \times (\text{number of 1's in the precedence matrix})}{n(n-1)} \qquad (3)$$

where the precedence matrix is an upper triangular binary matrix with (i, j)th entry equals to one if task j is a follower of task i on the precedence diagram, zero otherwise. Note that an F-Ratio of 1 corresponds to a chain precedence diagram whereas 0 corresponds to the bin-packing problem.

The task times of all thirty problems are generated from the binomial distribution ($n = 30$, $p = 0.25$). These parameters are also the choices of Leu *et al.* (1994) and Talbot *et al.* (1986). Finally, we choose the cycle time as 56, which is about twice the average of the maximum task times of the 30 problems.

We examine four DPA and GA parameter settings, namely DPC, warm-up period (WU), number of iterations (ITER), and population size (POPSIZE). DPC and warm-up period are the two DPA parameters. Number of iterations and population size are the two GA parameters included in the analysis.

The first factor, DPC, has four levels 0, 0.01, 0.02, and 0.03. DPC at 0 level corresponds to GA without DPA. As we increase the value of DPC from 0 to any other number (between 0 and 1) we turn on the DPA function.

The second factor is the warm-up period. This factor has five levels: 0, 25, 50, 75, and 500 iterations. DPA is applied with no warm-up period at the 0 level. We use 500 as the DPA level to observe the effects of a very long warm-up period.

The third factor is the number of iterations. Three levels are used: 500, 1000, and 2000. Finally, the fourth factor is the population size with four levels at 20, 30, 40, and 50. In the experiments, we keep the mutation rate at a fixed level to save from additional computation time. Based on pilot runs, we set the mutation rate to 0.05.

In the experiments, we take 10 replications of each problem at each combination of factor levels, by using the same set of 10 random seeds. Therefore, we solve 30 (problems) $\times$ 10 (replications) $\times$ 4 (DPC levels) $\times$ 5 (warm-up levels) $\times$ 3 (iteration levels) $\times$ 4 (population size levels) $=$ 72,000 problems.

5. Results of experiments

As given in Table 1, the effects of all four factors on CPU times are significant at the 5% level. For fitness scores, most of the factors are also significant, except for the warm-up period factor with 50% F-Ratio, and DPC and warm-up period factors with 90% F-Ratio.

The Bonferroni and Duncan groupings of the fitness scores are reported in Table 2. In general, these two methods yield the same ranking in all the experiments. DPA performs significantly better than the GA without DPA, (DPC at 0), in both the 10% and the 50% F-Ratio cases. When DPC is at the optimal level, the improvement of the average fitness scores with DPA compared to GA without DPA is 7.69% and 16.43% in the 50% and 10% F-Ratio cases, respectively. In the 90% F-Ratio case, GA with DPA does not perform significantly better than GA without DPA, but it is slightly better at all levels of the DPC.

DPC performs usually well at the level nearest to 0 level (i.e., 0.01). But the optimal value of this factor depends on F-Ratio and average task time compared to cycle time.

As can be noted in Table 2, there is a payoff between fitness score and CPU time as we change the value of DPC. In the 10% F-Ratio case, the improvement of the average fitness score is about 16% (8% in 50% F-ratio case) while the CPU time saving is more than 20% (23% in 50% F-ratio case) when DPC is at 0.01. However, when DPC is at 0.02 level, the improvement in the 10% F-ratio case decreases to 6% (1% in 50% F-ratio case) while the CPU time saving increases to 29% (31% in 50% F-ratio case). We do not observe this behavior in the 90% F-ratio case.

The effects of other factors on CPU time are as follows. CPU time increases as the number of iterations or the population size or the warm-up period increases. The effect of each level of these factors differs significantly from each other. We also observe that the performance improves significantly as the number of iterations increases at all levels of DPC. This observation was expected since the fitness score is not allowed to get worse than the value obtained at a prior iteration. The improvement in

Table 1. ANOVA results for fitness scores and CPU time

Source	*DF*	*Fitness scores*				*CPU time*			
F-Ratio = 10%		*Sum of sq*	*F value*	*Pr > F*	*Sig. at 0.05?*	*Sum of sq*	*F value*	*Pr > F*	*Sig. at 0.05?*
Model	167	14378.71	34.68	0.0001	yes	317886844.56	423.09	0.0001	yes
Error	23832	59159.74				107222212.01			
DPC	3	1966.58	261.07	0.0001	yes	39965797.16	2961.03	0.0001	yes
ITER	2	9772.55	1968.39	0.0001	yes	221475236.12	24613.36	0.0001	yes
WU	4	74.48	7.50	0.0001	yes	12618309.58	701.16	0.0001	yes
POPSIZE	3	793.68	106.58	0.0001	yes	14709022.85	1089.78	0.0001	yes
ITER*DPC	6	663.74	44.56	0.0001	yes	19417914.67	719.33	0.0001	yes
DPC*WU	12	305.00	10.24	0.0001	yes	4735933.37	87.72	0.0001	yes
DPC*POPSIZE	9	88.10	3.94	0.0001	yes	89602.23	2.21	0.0185	yes
ITER*WU	8	229.55	11.56	0.0001	yes	3341638.94	92.84	0.0001	yes
ITER*POPSIZE	6	283.70	19.05	0.0001	yes	172506.21	6.39	0.0001	yes
WU*POPSIZE	12	18.37	0.62	0.8302	no	93323.25	1.73	0.0544	no
DPC*WU*POPSIZE	36	31.24	0.35	0.9999	no	49551.79	0.31	1.0000	no
ITER*DPC*WU	24	113.32	1.90	0.0049	yes	1165622.78	10.79	0.0001	yes
ITER*DPC*POPSIZE	18	30.41	0.68	0.8340	no	26556.49	0.33	0.9966	no
ITER*WU*POPSIZE	24	7.99	0.13	1.0000	no	25829.11	0.24	1.0000	no
F-Ratio = 50%									
Model	167	2809.20	6.50	0.0001	yes	338013613.13	883.86	0.0001	yes
Error	23832	61721.60				54575204.98			
DPC	3	761.25	97.98	0.0001	yes	47259999.22	47974.24	0.0001	yes
ITER	2	1128.53	217.87	0.0001	yes	219721716.08	47974.24	0.0001	yes
WU	4	7.02	0.68	0.6074	no	22186867.76	2422.15	0.0001	yes
POPSIZE	3	165.46	21.30	0.0001	yes	10170984.90	1480.49	0.0001	yes
ITER*DPC	6	130.21	8.38	0.0001	yes	20175884.62	1468.41	0.0001	yes
DPC*WU	12	167.56	5.39	0.0001	yes	8385227.70	305.14	0.0001	yes
DPC*POPSIZE	9	65.94	2.83	0.0025	yes	37068.83	1.80	0.0631	no
ITER*WU	8	40.64	1.96	0.0471	yes	7002495.84	382.23	0.0001	yes
ITER*POPSIZE	6	183.56	11.81	0.0001	yes	57952.60	4.22	0.0003	yes
WU*POPSIZE	12	42.11	1.35	0.1797	no	205032.25	7.46	0.0001	yes
DPC*WU*POPSIZE	36	28.66	0.31	1.0000	no	106336.83	1.29	0.1144	no
ITER*DPC*WU	24	22.70	0.37	0.9981	no	2554821.89	46.49	0.0001	yes
ITER*DPC*POPSIZE	18	45.92	0.99	0.4735	no	45259.33	1.10	0.3464	no
ITER*WU*POPSIZE	24	19.63	0.32	0.9994	no	103965.27	1.89	0.0053	yes
F-Ratio = 90%									
Model	167	4940.09	1.35	0.0001	yes	472137987.51	3671.42	0.0001	yes
Error	23832	520551.62				18351793.51			
DPC	3	0.27	0.00	0.9996	no	1894134.57	819.92	0.0001	yes
ITER	2	1250.52	28.63	0.0001	yes	459903043.76	99999.99	0.0001	yes
WU	4	0.36	0.00	1.0000	no	654259.41	212.41	0.0001	yes
POPSIZE	3	3379.81	51.58	0.0001	yes	7903184.28	3421.08	0.0001	yes
ITER*DPC	6	0.06	0.00	1.0000	no	845397.93	182.98	0.0001	yes
DPC*WU	12	0.84	0.00	1.0000	no	467938.31	50.64	0.0001	yes
DPC*POPSIZE	9	0.75	0.00	1.0000	no	27598.34	3.98	0.0001	yes
ITER*WU	8	0.01	0.00	1.0000	no	229897.42	37.32	0.0001	yes
ITER*POPSIZE	6	305.90	2.33	0.0296	yes	6303.33	1.36	0.2246	no
WU*POPSIZE	12	0.50	0.00	1.0000	no	17151.22	1.86	0.0346	yes
DPC*WU*POPSIZE	36	0.64	0.00	1.0000	no	12427.69	0.045	0.9982	no
ITER*DPC*WU	24	0.02	0.00	1.0000	no	159493.27	8.63	0.0001	yes
ITER*DPC*POPSIZE	18	0.36	0.00	1.0000	no	11582.80	0.84	0.6591	no
ITER*WU*POPSIZE	24	0.06	0.00	1.0000	no	5572.18	0.30	0.9936	no

Table 2. Bonferroni and Duncan grouping of fitness scores and CPU time due to DPC

Fitness score				*CPU time*			
Bonferroni & Duncan	*Mean*	*N*	*DPC*	*Bonferroni & Duncan*	*Mean*	*N*	*DPC*
10% F-Ratio							
A	3.733	6000	0.03	A	3.289	6000	0
B	3.548	6000	0	B	2.605	6000	0.01
C	3.323	6000	0.02	C	2.340	6000	0.02
D	2.965	6000	0.01	D	2.241	6000	0.03
50% F-Ratio							
A	3.743	6000	0.03	A	3.246	6000	0
B	3.510	6000	0	B	2.510	6000	0.01
B	3.480	6000	0.02	C	2.235	6000	0.02
C	3.240	6000	0.01	D	2.096	6000	0.03
90% F-Ratio							
A	10.623	6000	0	A	3.237	6000	0
A	10.620	6000	0.03	B	3.167	6000	0.01
A	10.618	6000	0.01	C	3.103	6000	0.02
A	10.614	6000	0.02	D	2.996	6000	0.03

performance is logarithmic, i.e., the improvement gets less as the number of iterations increases.

Warm-up period factor is not significant in none of the F-Ratio levels. Hence, we drop out this factor (i.e., keep it fixed at the zero level) in our further experiments. We also note that this factor needs a different tuning at each level of the DPC. The performance improves at a decreasing rate with the number of iterations factor.

The population size factor can be tuned for obtaining the optimal performance of the algorithm. From the three sets of problems with 10%, 50%, and 90% F-ratios, we observed that a larger population size yields a better score on problems with higher F-Ratio (i.e. 50% and 90%). It may seem to be counter-intuitive, at the first sight, that smaller population sizes performed better than the larger ones on problems with the 10% F-Ratio. Our explanation for this observation stems from the fact that the search space is wider at low F-Ratios. Therefore, a large population cannot concentrate on local minimum search. The special recombination mechanism that is used in our GA is responsible for this finding, i.e. only one pair of chromosomes are selected for recombination at each iteration. Since including the best-fit chromosome in crossover is potentially more advantageous for the local minimum search than crossing over two other chromosomes, the performance deteriorates as the population size increases. In other words, the probability of selecting the best-fit chromosome for recombination in a large population is less than in a small population.

When the three sets of data are combined, we observed that F-Ratio is a significant factor on the overall performance of the system. The Bonferroni and Duncan grouping of the fitness scores due to F-

Table 3. Bonferroni and Duncan grouping of fitness scores due to F-Ratio

Bonferroni grouping	*Duncan grouping*	*Mean*	*N*	*F-Ratio*
A	A	10.619	24,000	90%
B	B	3.493	24,000	50%
C	C	3.392	24,000	10%

Ratio is presented in Table 3. Although the average task time of each set of problems is approximately the same (i.e., they are generated by the same random generator), the fitness scores increase exponentially as the F-Ratio increases, since the increase in the number of precedence relations reduces the allocation alternatives of the tasks and leading to an increase in the required number of stations.

In conclusion, the DPA procedure achieved a significant amount of CPU time saving. Even though some deterioration in the performance is expected due to DPA, we have surprisingly achieved some improvement. In other words, we have obtained a better performance with DPA than the traditional application of GA (without DPA), while also saving from the CPU time. This counter-intuitive result can be explained as follows. The stations that are frozen by DPA already have station times that minimize the fitness function, as explained earlier. Hence, by freezing some of the tasks without straying too much from optimal balancing, the GA concentrates more on the remaining tasks. If we did not freeze the stations that satisfy the DP criteria, the mutation and crossover mechanisms would waste time by working on these already balanced stations as well, instead of focusing on the poorly balanced stations. Therefore, given the same number of iterations, a GA with DPA is able to work (try alternative combinations) on balancing the poorly balanced stations more than a GA without DPA. Consequently, we achieved significant performance improvement by DPA.

Another interesting observation is that the improvement effect of DPA decreases as the F-Ratio increases, i.e., as the search space gets narrower. The reason is that the possibility of partitioning at the same level of DPC decreases due to small number of feasible solutions resulting from the large number of precedence relationships. Even if the GA is allowed to focus on the poorly balanced stations with DPA, it is less likely to lead to an improved result since GA without DPA can perform a sufficient search in a narrow search space.

6. Restructuring elitism by simulated annealing

Replacing a parent with an offspring only if the offspring is better than the parent is called *elitism.* There is an analogy between the idea of elitism in GAs and local search algorithms, a move to a neighbor point is made only if the solution is improved. In this section, we restructure the elitism rule of our GA with the simulated annealing (SA) methodology.

SA is a well-known global search algorithm in which moves to poorer solutions are allowed with a certain probability. We refer the interested readers to articles by Johnson *et al.* (1989, 1991) on SA.

6.1. *Integration of SA to elitism*

The problem specific decision elements of SA are replaced by GA decision elements in our application. The initial solution is the best-fit chromosome of the initial population, neighborhood generation is simply the crossover and mutation mechanisms, and evaluation of ∇c is the difference between the fitness scores of the offspring and its parent. In case the offspring's fitness score is larger than its parent's, we calculate ∇c, and then evaluate the probability function, $P(x) = \min(1, \exp(-\frac{\nabla c}{T}))$. Temperature (T) is decreased exponentially as $T_{k+1} = T_k \times \alpha$, where k is the iteration number, and α is the scaling factor smaller than 1 and usually very close to 1, i.e., 0.98. Hence, $T_k = T_0 \times \alpha^k$, at the kth iteration. We do not explicitly define a stopping criterion other than the 0.01 limit for T. We keep the iteration number fixed at 500, but $P(x)$ starts to take values that are almost zero after T reaches the 0.01 limit, hence this limit can be thought as the stopping criterion of SA where strict elitism takes over again. With the above stopping criterion, $P(x)$ reaches approximately zero at different iteration numbers due to different α levels. In our experimental setup, we used 7 different levels of α, 0, 0.8, 0.95, 0.96, 0.97, 0.98, and 1. The level 0 means ''strict elitism,'' i.e., no SA, and the level 1 means ''no elitism'' where our crossover mechanism (neighborhood generation mechanism) turns out to be a random search mechanism instead of a local-optimum seeking mechanism.

6.2. *Experimentation*

Our experimental design again consists of the same 30 problems with 10 replications, 4 population size factors (20, 30, 40, 50), and 7 α levels (0, 0.80, 0.95, 0.96, 0.97, 0.98, 1), resulting in $30 \times 10 \times 4 \times 7 = 8,400$ instances.

The Anova results are given in Table 4. We observe that α levels are significantly different from each

Table 4. ANOVA results for fitness scores

Fitness scores					
Source	*DF*	*Sum of squares*	*F value*	*Pr > F*	*Significant at 0.05?*
Number of Iterations = 500					
Model	41	64954.54	171.63	0.0001	yes
Error	8358	77151.19			
ALPHA	6	801.64	14.47	0.0001	yes
F-RATIO	2	63714.08	3421.91	0.0001	yes
POPSIZE	3	90.48	3.27	0.0204	yes
F-RATIO*ALPHA	12	801.65	7.24	0.0001	yes
POPSIZE*ALPHA	18	86.68	0.52	0.9499	no
Number of Iterations = 1000					
Model	41	72736.44	220.21	0.0001	yes
Error	8358	67334.30			
ALPHA	6	1108.85	22.94	0.0001	yes
F-RATIO	2	70000.85	4344.50	0.0001	yes
POPSIZE	3	273.72	11.33	0.0001	yes
F-RATIO*ALPHA	12	1292.04	13.36	0.0001	yes
POPSIZE*ALPHA	18	60.97	0.42	0.9844	no

Table 5. Bonferroni and Duncan grouping of fitness scores due to α, F-Ratio, population size

Bonferroni grouping Iter = *500*	*Duncan grouping*	*Mean*	*N*	α	*Bonferroni grouping Iter* = *1000*	*Duncan grouping*	*Mean*	*N*	α
A	A	7.368	1200	1	A	A	6.925	1200	1
B A	B	7.018	1200	0.98	B A	B	6.608	1200	0.98
B C	C	6.712	1200	0.97	B C	C	6.315	1200	0.97
C	C	6.596	1200	0.95	D C	C	6.209	1200	0.96
C	C	6.585	1200	0.96	D C E	C	6.123	1200	0.95
C	C	6.494	1200	0.8	D E	D	5.876	1200	0
C	C	6.440	1200	0	E	D	5.825	1200	0.8
				F-Ratio					F-Ratio
A	A	10.609	2800	90 %	A	A	10.351	2800	90 %
B	B	5.091	2800	10 %	B	B	4.271	2800	50 %
C	C	4.534	2800	50 %	B	B	4.184	2800	10 %
				Pop size					Pop size
A	A	6.900	2100	20	A	A	6.554	2100	20
B A	B A	6.762	2100	30	B	B	6.287	2100	30
B A	B	6.700	2100	50	B	C B	6.153	2100	50
B	B	6.616	2100	40	B	C	6.080	2100	40

other. We ranked the factors by both Bonferroni and Duncan methods (Table 5). According to Bonferroni, the α levels are not significantly different from each other, but Duncan test ranks 1 and 0.98 levels separately from the other levels. We should point out here that Bonferroni is a conservative test. Thus, we conclude that elitism is better than the no elitism case. Then we increase the number of iterations factor to 1000 to see if we can observe a significant difference in the Bonferroni ranking as well. (We change the initial temperature from 1000 to 10,000,000 in order to avoid early cooling, as we change the number of iterations from 500 to 1000). We observed again that the Bonferroni ranking of the α levels do not show any significant difference, but the Duncan test ranks the levels in four groups instead of three.

Later, we enlarged our experimental design by including three different DPC levels, i.e., 0.01, 0.02, and 0.03, in addition to the other factors. In this case, we observe that the combined effect of α and DPC is significant at 0.05 level, but overlapping of the levels of α is observed according to the Bonferroni grouping.

Although we could not achieve any significant improvement by relaxing the elitism rule, we observe that strict elitism ($\alpha = 0$) is significantly better than no elitism ($\alpha = 1$). Considering that our reproduction mechanism is a special one which is different from the traditional approach, i.e. only one or two chromosomes are replaced by new offsprings, we claim that elitism should be used in order to obtain a better performance with this kind of a reproduction mechanism.

7. Comparison of the proposed GA with heuristics

In this section we compare the proposed GA with Leu *et al.*'s (1994) GA on the Kilbridge-Wester's (1961) 45-task ALB problem and with Baybars' (1986) heuristic and other traditional heuristics on Tonge's (1961) 70-task problem.

7.1. *Comparison with leu et al.'s GA (1994)*

Leu *et al.* (1994) solved Kilbridge-Wester's (1961) problem by their GA and compared it with five other heuristics that are also available in the QS software package (Chang and Sullivan, 1991). These five non-GA heuristics are single-pass procedures accompanied by heuristic rules to break ties.

The cycle time of the original problem is 55, but Leu *et al.* (1994) slightly change this value to 56 to observe the sensitivity of non-GA heuristics to changes in problem constraints. The authors also compare the five heuristics with their GA using four different measures, (i) mean-squared idle time, (ii) square root of mean squared idle time, (iii) efficiency (utilization), and (iv) maximum station time. If the maximum station time is less than the given cycle time, then it becomes the new cycle time, i.e., the cycle time is reduced. The other three measures are already explained in Section 2. We evaluate the performance of our GA in terms of these measures as well.

We solved the Kilbridge-Wester problem by the proposed GA using 1200 different factor level combinations. The factors used and their levels are: DPC (0, 0.01, 0.03, 0.05), population size (20, 50), cooling rate (0, 0.95, 0.97, 0.99, 1), mutation rate (0.02, 0.05, 0.1), and 10 random seeds. In the experiments, we set the number of iterations = 500, warm-up period = 0.

As shown in Table 6, the proposed GA finds the optimal number of stations, outperforming Leu *et al.*'s GA (1994) and the other heuristics. The solution that

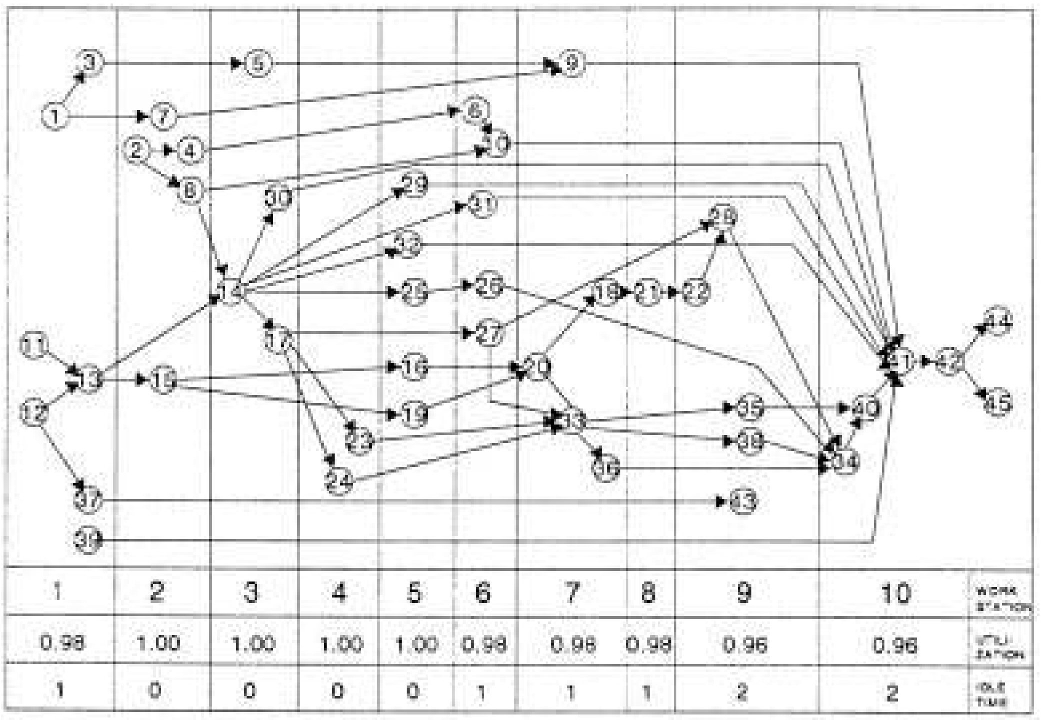
Table 6. Comparison of non-GA heuristics, Leu *et al.*'s GA and the proposed GA

Solution method	*Mean-squared idle time*	*Sqr root (mean-sqrd idle time)*	*Efficiency*	*Maximum workload*
Heuristic 1	239.64	15.48	0.8961	56
Heuristic 2	239.27	15.47	0.8961	56
Heuristic 3	67.45	8.21	0.8961	56
Heuristic 4	124.91	11.17	0.8961	56
Heuristic 5	239.64	15.48	0.8961	56
Leu *et al.*'s GA	51.81	7.20	0.8961	55
Proposed GA	1.20	1.10	0.9855	56

Table 7. Factor levels at which the optimal solution is found

No	*DPC*	*Population size*	*Random seed*	*Cooling rate*	*Mutation rate*	*CPU time*	*Mean sqrd idle time*
1	0.03	20	14567	0.97	0.05	0.76	1.40
2	0.03	20	97663	0.99	0.02	0.76	1.40
3	0.03	20	97665	0.99	0.05	0.82	1.20
4	0.03	20	77943	0.99	0.10	1.04	1.40
5	0.03	20	47729	0.99	0.10	0.93	1.40
6	0.03	20	77943	1.00	0.10	1.04	1.40
7	0.03	50	84521	0.97	0.10	1.37	1.20
8	0.03	50	76421	0.97	0.10	1.59	1.20
9	0.03	50	60013	0.99	0.10	1.70	1.40
10	0.03	50	14567	1.00	0.02	1.53	1.40
11	0.05	20	14567	0.97	0.05	0.76	1.40
12	0.05	20	77943	0.99	0.10	1.04	1.40
13	0.05	20	47729	0.99	0.10	0.93	1.40

provides the optimal number of stations was found at 13 different factor level combinations. These factor levels are presented in Table 7 to demonstrate the positive effect of DPA and the restructured elitism rule. As can be observed in Table 7 the optimal number of stations (i.e., 10) is found by the proposed GA only when DPA is activated (i.e., DPC ≠ 0) together with the restructured elitism (i.e., cooling rate ≠ 0). The two different mean squared idle time measures in this table indicate that there are two alternative solutions with 10 stations. The solution with the lowest mean squared idle time is given in Fig. 4.



1	2	3	4	5	6	7	8	9	10	WORK STATION
0.98	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.96	0.96	UTILI ZATION
1	0	0	0	0	1	1	1	2	2	IDLE TIME

Fig. 4. The proposed GA solution for the Kilbridge-Wester 45-Task problem.

7.2. *Comparison with Baybars' LBHA-1 (1986)*

Baybars (1986) solved Tonge's (1961) 70-task problem with a heuristic called LBHA-1. Tonge's (1961) problem is a real life application that comes from the electronics industry. Since 1965, numerous attempts have been made to solve the Tonge (1961) problem for 13 different cycle times ranging from 83 to 364.

We solve Tonge's (1961) problem for the 13 cycle times with the proposed GA. Our algorithm have five parameters, i.e., DPC, number of iterations, cooling rate, mutation rate, and population size, that need to be optimized for each problem. First, we experimented the effect of each parameter on the performance for minimum cycle time and fixed the number of iterations factor to 500 because we did not observe any significant improvement at higher levels. We observed that the best performing level of DPC is 0.05, hence we eliminated all other levels except the 0 level, which we kept to observe the GA without DPA performance. The mutation parameter's levels are 0.01, 0.03, 0.05, 0.10, 0.20, and the cooling rate parameter takes 0, 0.95, 0.97, 0.99, 1 values for all versions. Hence, we solved the problem 500 times for each of the 13 cycle times, i.e., 5 (mutation) $\times$ 5 (cooling rate) $\times$ 2 (DPC) $\times$ 10 (seeds or replications) $= 500$. We took the best solution, i.e., the minimum number of stations, among these 500 solutions as our solution to the problem in Table 8. The optimal solutions to these problems as well as the results of previous studies that have attempted to solve this problem are also given in Table 8.

It can be observed from Table 8 that the proposed GA performs better than all heuristics except Nevins' (1972) and Baybars' (1986). However, there is no significant difference between the performance of Nevins' (1972) or Baybars' (1986) heuristics and the proposed GA. The proposed GA solutions match those of Baybars except for four cases in which we exceed the optimal solution by only one and for one case in which we find the optimal solution while Baybars' LBHA-1 does not. Considering that the proposed GA finds five of the thirteen optimal solutions and finds solutions to the other cases with only one extra station than the optimum, it performs quite well on the Tonge's (1961) problem.

Although GAs are applicable to any kind of ALB problem regardless of the F-Ratio, we observe that they perform worse in problems with high F-Ratio, as in Tonge's (1961) problem with 59.42% F-Ratio. If the number of precedence relations increases, the possibility of generating offsprings that are better than their parents decreases. In such a case, another crossover operator that provides more substantial changes on the parents' genes may be used instead of a moderate crossover operator like ours. The purpose of the two point crossover, used in our algorithm, is to conduct a neighborhood search by keeping the head and the tail of each offspring the same as its parent. The offspring should be close in fitness to its parent since only its middle genes change. Conversely, a one

Table 8. Comparison of seven methods on the 70-task problem of Tonge (1961) in terms of number of stations

Cycle time	*Optimal solution*	*Moodie and Young (1965)*	*MIF*	*Tonge (1965) RC*	*BPC*	*Nevins (1972)*	*Baybars (1986)*	*Proposed GA*
83	47	48	50	50	49	47	47	48
86	46	47	47	48	47	46	46	46
89	43	44	45	46	44	43	43	44
92	?	43	43	44	43	42	42	43
95	40	42	43	43	41	40	40	41
170	22	24	24	24	23	23	23	23
173	22	24	24	24	23	22	23	23
176	22	22	24	23	22	22	23	22
179	21	22	23	23	21	21	22	22
182	21	22	23	22	21	21	22	22
346	11	11	11	12	11	11	11	11
349	11	11	11	11	11	11	11	11
364	11	11	11	11	11	11	11	11

point crossover would change, on the average, half of the entire chromosome of each offspring, and such a change could be too drastic and might move the offspring out of the local search neighborhood. Similarly, more-than-two-point crossover could result in changes in fitness that are either too small or too large depending on how the swapping is done. Hence, a one point crossover might achieve better results compared to our two point crossover operator for problems with high F-Ratios.

Note also that Baybars' LBHA-1 consists of reduction phases that reduce the problem size by eliminating tasks, determining mutually exclusive task sets, and decomposing the network, while no reduction phases are applied to the problem prior to our GAs. It has been shown by Leu *et al.* (1994) that starting the GA with a better initial population significantly improves the solution quality.

8. Discussion

In this paper, we developed a new GA to solve the deterministic and single-model ALB problem by utilizing the intrinsic characteristics of the problem. We showed that the chromosome structure of GAs can be changed dynamically to provide an efficient search in the ALB solution space. In particular, we reduced the chromosome size during the search procedure that led to improvements both in the solution quality and computation times. By the same token, this new dynamic chromosome structure is novel concept that can be applied to other GA applications. Furthermore, we constructed a new elitism structure adopted from the concept of SA. We observed that this new elitism structure contributes significantly to the performance of the GA. In fact, the results of extensive computational experiments indicated that the proposed GA approach outperforms the well-known heuristics in the literature.

ALB problem is a frequently-addressed problem both during the design and life-cycle of the product, it is solved in several stages due to product model changes and unbalancing caused by uncontrollable factors such as learning phenomenon. Considering the significant monetary consequences of having a poor design, it is extremely important to utilize an efficient and practical approach for managers responsible for the design and control of assembly lines. In this paper, we developed such a novel and efficient GA approach to solve the real size ALB problems.

There are several future research directions that can originate from this study. First, a static partitioning procedure developed in this study can be applied to divide large ALB problems (more than 100 tasks) into smaller subproblems to improve the solution time. Secondly, the proposed DPA approach can be extended to freeze not only the tasks of the stations at the beginning or at the end of the line but also of the other stations. One can expect that such a revision would improve the performance of the GA especially for large size problems. Thirdly, in this study we considered only the single model and deterministic ALB problem, however the scope of the study can be extended to multi/mixed model and/or stochastic cases as well. An immediate extension would be done to the U-type assembly line systems which are often encountered in practice. Finally, the DPA and elitism structure proposed in this study can be enhanced with different crossover and mutation operators, and coding representations.

References

Anderson, E. J. and Ferris, M. C. (1994) Genetic algorithms for combinatorial optimization: the assembly line balancing problem. *ORSA Journal on Computing*, **6**, 161–173.

Baybars, I. (1986) An efficient heuristic method for the simple assembly line balancing problem. *International Journal of Production Research*, **24**, 149–166.

Bowman, E. H. (1960) Assembly line balancing by linear programming. *Operations Research*, **8**, 385–389.

Chang, Y.- L. and Sullivan, R. S. (1991) *QS Quant Systems, version 2*, Englewood Cliffs, NJ.

Dar-El, E. M. (1973) MALB — A heuristic technique for balancing large scale single-model assembly lines. *AIIE Transactions*, **5**, 343–356.

Dar-El, E. M. and Rubinovitch, Y. (1979) MUST — A multiple solutions technique for balancing single model assembly lines. *Management Science*, **25**, 1105–1114.

Davis, L. (1985) Applying algorithms to epistatic domains. In *Proceedings International Joint Conference on Artificial Intelligence*.

Ghosh, S. and Gagnon, R. J. (1989) A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. *International Journal of Production Research*, **27**, 637–670.

Goldberg, D. E. (1989) *Genetic algorithms in search,* *optimization and machine learning*, Addison-Wesley Publishing Company Inc., Reading, MA.

Held, M., Karp, R. M. and Shareshian, R. (1963) Assembly line balancing — dynamic programming with precedence constraints. *Operations Research*, **11**, 442–459.

Helgeson, W. B., Salveson, M. E. and Smith, W. W. (1954) *How to balance an assembly line*, Technical Report, Carr Press, New Caraan, Conn.

Holland, J. H. (1975) *Adaptation in natural and artifical systems*, The University of Michigan Press, Ann Arbor, MI.

Jackson, J. R. (1956) A computing procedure for a line balancing problem. *Management Science*, **2**, 261–271.

Johnson, D. S., Aragon, C. R., McGeoch, L. A. and Scheuon, C. (1989) Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. *Operations Research*, **37**, 865–892.

Johnson, D. S., Aragon, C. R., McGeoch, L. A. and Scheuon, C. (1991) Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. *Operations Research*, **39**, 378–406.

Johnson, R. V. (1981) Assembly line balancing algorithms: computational comparisons. *International Journal of Production Research*, **19**, 277–287.

Kilbridge, M. D. and Wester, L. (1961) A heuristic method of assembly line balancing. *The Journal of Industrial Engineering*, **12**, 292–298.

Klein, M. (1963) On assembly line balancing. *Operations Research*, **11**, 274–281.

Leu, Y. Y., Matheson, L. A. and Rees, L. P. (1994) Assembly line balancing using genetic algorithms with heuristic-generated initial populations and multiple evaluation criteria. *Decision Sciences*, **25**, 581–606.

Moodie, C. L. and Young, H. H. (1965) A heuristic method of assembly line balancing for assumptions of constant or variable work element times. *Journal of Industrial Engineering*, **16**, 23–29.

Nevins, A. J. (1972) Assembly line balancing using best bud search. *Management Science*, **18**, 529–539.

Patterson, J. H. and Albracht, J. J. (1975) Assembly-line balancing: zero-one programming with fibonacci search. *Operations Research*, **23**, 166–172.

Schrage, L. and Baker, K. R. (1978) Dynamic programming solution of sequencing problems with precedence constraints. *Operations Research*, **26**, 444–449.

Suresh, G., Vinod, V. V. and Sahu, S. (1996) A genetic algorithm for assembly line balancing. *Production Planning and Control*, **7**, 38–46.

Talbot, F. B. and Patterson, J. H. (1984) An integer programming algorithm with network cuts for solving the assembly line balancing problem. *Management Science*, **30**, 85–99.

Talbot, F. B., Patterson, J. H. and Gehrlein, W. V. (1986) A comparative evaluation of heuristic line balancing techniques. *Management Science*, **32**, 430–454.

Tonge, F. M. (1961) *A heuristic program of assembly line balancing*, Englewood Cliffs, NJ.

Tonge, F. M. (1965) Assembly line balancing using probabilistic combinations of heuristics. *Management Science*, **11**, 727–735.

Wee, T. S. and Magazine, M. (1981) *An efficient branch and bound algorithm for an assembly line balancing problem — part I: minimize the number of work stations*, University of Waterloo, Ontario, Canada.

Whitely, D. and Kauth, J. (1988) GENITOR A different genetic algorithm. In *Rocky Mountain Conference on Artificial Intelligence*.