ELSEVIER

Int. J. Production Economics 41 (1995) 343–354

international journal of
production
economics

Genetic algorithm for assembly line balancing

J. Rubinovitz*, G. Levitin

Faculty of Industrial Engineering and Management, Technion, Haifa 32000, Israel

Accepted for publication 18 May 1995

Abstract

Research on single-model assembly line balancing has produced several good algorithms for solving large problems. The majority of these algorithms generate just one solution to the problem, whereas the real line design faces the need to investigate alternative solutions, where preferences for work allocation to stations are considered, or constraints other than technological precedence are taken into account. The MUST algorithm suggested by Dar-El and Rubinovitch is one of the few algorithms that provides such diversity of solutions. Heuristics enable MUST to solve relatively large line-balancing problems. However, when the initial problem has relatively few constraints, the time and memory requirements needed by MUST may become excessive.

This paper describes the development and testing of a Genetic Algorithm for the generation of multiple solutions to the assembly line balancing (ALB) problem. The results are compared with MUST results for different classes of problems.

Good results are achieved by combining the genetic approach with a simple local optimization procedure. This procedure performs much faster than MUST for problems with large number of stations and high flexibility ratio. Different crossover and mutation procedures are tested and evaluated, in order to recommend these which are most effective for solving ALB problems.

Keywords: Assembly line balancing; Genetic algorithms

1. Introduction

The ALB (assembly line balancing problem) is how to group the assembly activities, which have to be performed in an assembly task, into workstations, so that the total assembly time required at each workstation is approximately the same. The cycle time of the assembly line is determined by the workstation with maximum total assembly time. Two formulation types are possible for this problem [1]: type I attempts to minimize the number of stations for a required cycle time, while type II formulation attempts to minimize the cycle time for a given number of stations. Using formulation II, the problem is how to group Na basic activities which compose the assembly task into Nst workstations so as to achieve the minimal cycle time. The grouping of activities into stations must be done without violation of precedence relations between the activities.

Two measures are commonly used to characterize the ALB problem: the flexibility ratio (F-ratio), and the work-element to station ratio (WEST-ratio) [2].

* Corresponding author.

SSDI 0925-5273(95)00059-3

The flexibility ratio measures the precedence relationship between work elements in the task. The precedence can be indicated by cells p_{ij} above the main diagonal of a rectangular matrix $\boldsymbol{P}$, where $p_{ij} = 1$ if work-element i precedes work-element j, and 0 otherwise. If H is the number of zero cells above the diagonal, than for an assembly task with Na work elements the F-ratio is defined as

$$F\text{-ratio} = \frac{2H}{\text{Na}(\text{Na} - 1)}.$$

The F-ratio gives an indication of the order strength of the assembly task: the higher the F-ratio, the fewer the precedence constraints and the greater the flexibility in generating different feasible station assignments.

The WEST-ratio is the average number of work-elements (activities) to station, i.e. WEST-ratio = Na/Nst.

At low values of WEST-ratio (Na $\geqslant$ Nst) good line balances are more difficult to achieve than at high values of WEST-ratio (Na $\gg$ Nst) for which a much larger number of feasible sequences and good station assignments exists.

The majority of algorithms suggested for solution of the ALB problem generate just one solution [3]. However, the real assembly line design needs to investigate alternative solutions, where preferences for work allocation to stations are considered, or constraints other than technological precedence are taken into account. The MUST algorithm suggested by Dar-El and Rubinovitch [4] is aimed to provide such necessary diversity of solutions. It employs a breadth-first search which is capable to generate all optimal solutions for small problems. For large problems MUST uses heuristic rules to limit the search and generate a subset of equal quality solutions. MUST is a very effective algorithm for problems in which precedence constraints or other constraint types (such as zoning or tooling) limit the search space. However, for large problems with a high F-ratio and a high WEST-ratio the breadth-first search results in an exponential growth in the computing time and memory resources required. To solve such problems, the MUST algorithm has to limit the search space by applying heuristics which may affect the quality of solutions reached.

Genetic algorithms (GA) have been applied successfully to solve complex combinatorial search problems in different application domains [5]. The family of genetic algorithms is based on a simple principle of evolutionary search in the space of solutions. The main characteristic of genetic algorithms is their robustness, which implies high independence between the search process and the problem complexity or size. GAs need only a simple evaluation function to estimate the quality of candidate solutions. The procedures of solution quality evaluation may be easily changed or modified, providing a desirable flexibility to consider all elements and factors of real assembly lines design and balancing.

In this paper an evaluation of the prospect of GA application to the ALB problem is made and the main features of such an algorithm are investigated. The performance of the GA is compared with MUST for a variety of problems. In addition, the GA-based ALB algorithm allows to balance a line where task times are station dependent. This is different to the traditional ALB methods which assume a constant task (work-element) time, regardless of station assignment. In reality, when using manual assembly lines, task times may be affected by the difference of worker skill and productivity at each station. In automated assembly lines, task times may be different at each station due to different equipment capabilities and performance.

2. Genetic algorithm

Unlike various constructive procedures which use sophisticated heuristics to obtain a good single solution, the genetic algorithms are dealing with a set of solutions and tend to manipulate each solution in the simplest way.

The basic steps of GA (in this work we have used the GENITOR-like modification of GA [6]) are:

(G1) Generation of initial population of randomly constructed solutions (a solution is represented by a problem-specific structure of characters or bits).

(G2) Random selection of two solutions and generation of new solutions using crossover procedures which are supposed to provide inheritance

of some basic properties of parent structures by the offsprings.

(G3) Mutation of child structures with probability p_m, which implies exchange of elements between two randomly selected positions in a structure.

(G4) Decoding/evaluation of the child structures to obtain the objective function values.

(G5) Selection procedure including comparison of child solutions with the worst solutions in the population and replacement of the worst solution by the new one if it is better. If after selection the population contains equal structures, the redundant structure is being removed and the population size is decreased.

(G6) Termination of the algorithm after repetition of steps (G2)–(G5) R times, where R is an initially specified parameter.

A comprehensive survey of the genetic algorithms theory, including construction of crossover, mutation and selection procedures, was provided by Goldberg [5].

The decoding/evaluation procedure (step G4) is the only one which is tightly connected with the nature of a problem being solved. This step has to include transformation operators (if necessary) and some means for the evaluation of solution quality. Incorporating some local optimization procedures into the transformation procedure may also significantly improve the performance of GA [7]. Some optimization methods may also be implemented in the stage of initial population generation (step G1). For instance, such an initial set of solutions may be generated using a set of simple single pass solution methods for ALB [3, 2]. For the sake of purity of GA efficiency evaluation we did not include such procedures in step G1, and the initial set of solutions was generated at random.

Solutions in genetic algorithms are represented by strings of different types ("genomes"). The choice of solution representation (structure) in GA is related to the nature of the problem. This representation also affects the method of transformation and evaluation. The most natural representation of ALB problem solution is an integer vector of activity numbers ordered according to the sequence of their execution and a second vector of pointers to the position of the first activity for each station. This vector of pointers divides the vector of activities into Nst stations. A sample solution representation for a small problem is shown in Fig. 1.

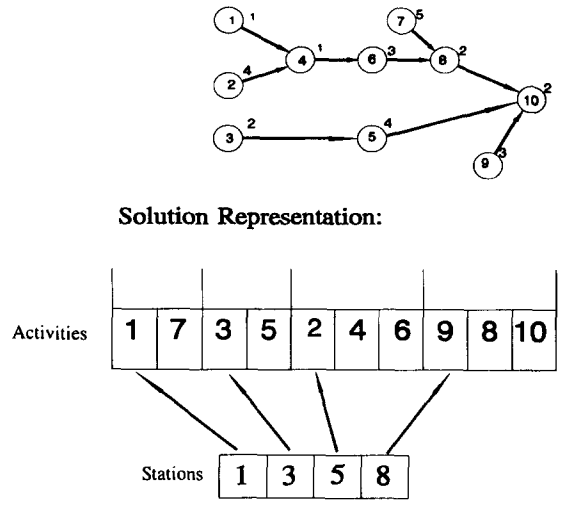

Fig. 1. A sample solution representation for a small problem.

It should be noted that such representation may cause situations in which equal solutions are represented by different vectors, because of feasible permutations of the order of activities within stations (for instance 17 or 71 for the first station indicated in Fig. 1). So a special procedure is necessary to check the equality of solutions.

3. Decoding/evaluation procedures for the ALB problem

A formal description of the decoding/evaluation procedures implemented for the ALB problem is given in this section. An illustrative example of these procedures is presented in Fig. 2. Before presenting these procedures, let us introduce the following notation:

- $\boldsymbol{P}$ precedence matrix in which each element p_{ij} is 1 if activity i immediately precedes activity j and 0 otherwise
- t_{kj} the time of jth activity execution on station k
- Y_j the set of immediate predecessors of activity j
- $s(j)$ the number of a station to which activity j is assigned
- T_k total execution time for station k

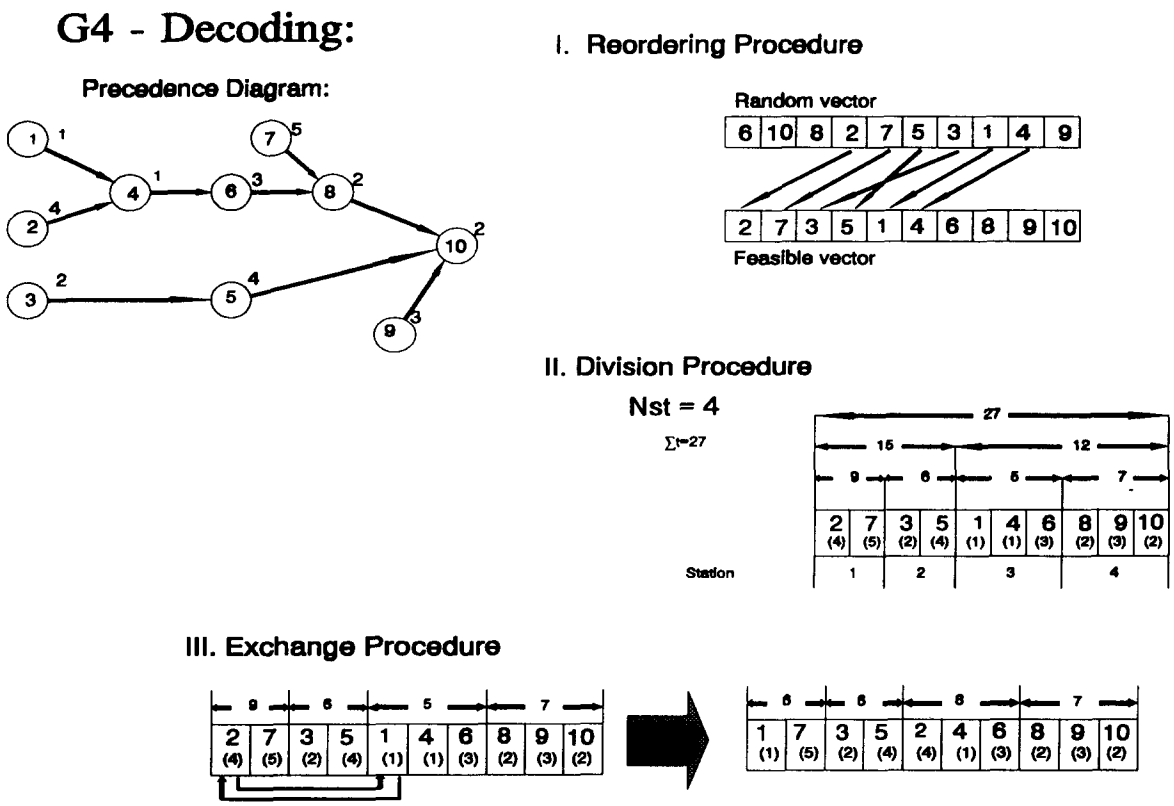

Fig. 2. An illustrative example of decoding/evaluation procedures (step G4 of GA) for ALB.

$v^{\sim}$ initial integer vector of activity numbers (randomly generated or obtained in the genetic process)

v integer vector after transformation (feasible solution representation)

3.1. Reordering procedure

The objective of this procedure is to transform an arbitrary vector $v^{\sim}$ of activity numbers into a sequence v which is feasible according to precedence relations.

Let us introduce a logical function $W_j(k)$ which returns "true" if activity j can be moved from station $s(j)$ to station k and "false" otherwise. $W_j(k)$ is false in two cases: (a) j is currently assigned to a station $s(j)$ which follows station k, and all the immediate predecessors Y_j of j are assigned to stations which follow station k; (b) j is currently assigned to a station $s(j)$ which precedes station k, and all the immediate followers of j are assigned to stations which precede station k. Using formal notation,

$$W_j(k) = \text{"false" if}$$

$$s(j) > k \quad \text{and} \quad \exists i \in Y_j, s(i) > k$$

or

$$s(j) < k \quad \text{and} \quad \exists i: j \in Y_i, s(i) < k,$$

$$W_j(k) = \text{true otherwise.}$$

The reordering procedure consists of the following steps:

(R1) For all activities $1 \leqslant j \leqslant \text{Na}$ assign $s(j) = 2$; $i = 1$.

(R2) Find the least m ($m \leqslant \text{Na}$):

$$s(v^{\sim}(m)) = 2 \quad \text{and} \quad W_{v^{\sim}(m)}(1) = \text{true}.$$

(R3) If such m does not exist end of procedure.

Else: $v(i) \leftarrow v^{\sim}(m)$;
$i \leftarrow i + 1$;
$s(v^{\sim}(m)) \leftarrow 1$;
return to (R2).

An example demonstrating how the reordering procedure works and generates feasible sequences is given in Fig. 2(I).

3.2. Division procedure

This procedure aims to divide the vector v of all the activities between a given number of stations Nst, without changing the feasible v sequence that was achieved by the re-ordering procedure. In order to minimize the resulting cycle time of the assembly line, the vector v has to be divided into $M = \text{Nst}$ parts so that a maximum possible equality between the execution times at all stations is achieved. The following recursive division procedure was developed to achieve this goal.

Let us first divide the total vector v (i.e. the set of its Na elements from position pl = 1 to position pr = Na) into two parts, trying to reach a ratio of H/Q between their execution times, where $H = [M/2]$ and $Q = M - H$. To do this, one has to find a position i (pl $\leqslant i \leqslant$ pr) in such a way that the ratio

$$\sum_{j=\text{pl}}^{i} t_{v(j)} \Big/ \sum_{j=i+1}^{\text{pr}} t_{v(j)}$$

is as close as possible to the ratio H/Q. Thus we have to minimize a function

$$d(i) = \left| Q \sum_{j=\text{pl}}^{i} t_{v(j)} - H \sum_{j=i+1}^{\text{pr}} t_{v(j)} \right|.$$

Finding such i we obtain two parts (pl = 1; pr = i and pl = $i + 1$; pr = Na) which we have to divide into $M = H$ and $M = Q$ parts, respectively, using the same procedure recursively until $M = 1$. In this last case the total execution time for each station is being calculated and the boundary positions pl and pr of each station are defined.

An example demonstrating how the division procedure works for $M = \text{Nst} = 4$ is given in Fig. 2(II).

3.3. Exchange procedure

The exchange procedure is a local optimization procedure, which attempts to reduce the cycle time of the assembly line by exchange of two activities from two different stations. Let us consider two stations r and q with total execution times T_r and $T_q (T_r > T_q)$. If exchange of activities i ($s(i) = r$) and j ($s(j) = q$) is feasible (i.e it does not violate precedence constraints), the new execution times after such exchange is

$$T_r^* = T_r - t_{ri} + t_{rj},$$

$$T_q^* = T_q - t_{qj} + t_{qi}.$$

The exchange is worthwhile if

$$\max\{T_r^*, T_q^*\} < T_r.$$

From these expressions one can derive the condition of exchange:

$$t_{rj} < t_{ri} \quad \text{and} \quad T_q - T_r < t_{qj} - t_{qi}. \tag{1}$$

To reduce the cycle time of an assembly line, one has to consider the time of most loaded station as T_r.

The exchange procedure is as follows:

(E1) Rank all stations in order of total execution times.

(E2) For most loaded station r and consequently for other stations $1 \leqslant q \leqslant \text{Nst}$, $q \neq r$ (beginning from the least loaded) look for a pair of activities (i, j): $s(i) = r$, $s(j) = q$ which satisfies condition (1), and the conditions for preserving precedence constraints:

$$W_i(q) = W_j(r) = \text{true};$$

$$p_{ij} = p_{ji} = 0.$$

(E3) If such pair of activities is found, perform the exchange, recalculate execution times T_r and T_q and return to (E1). If such wanted pair of activities does not exist for all possible q, i and j, terminate the procedure.

4. Computational results

An extensive set of tests was performed to evaluate the applicability and the effectiveness of the genetic algorithm developed in this work for the solution of assembly line-balancing problems. The experimental data set was composed of different assembly networks (3 with Na = 107 and 3 with Na = 142) with F-ratio (flexibility ratio, a measure of the assembly task precedence constraints order-strength) values of 0.1, 0.4 and 0.8 correspondingly. Each problem was solved for 6 Nst values from 10 to 35 with an increment of 5. So the total number of problems solved was 36.

The tests performed were aimed to answer the following questions related to the applicability and effectiveness of GA for ALB:

1. Is the GA approach applicable at all for the ALB problem, which requires complicated transformations of an initial integer vector $v^{\sim}$ in order to obtain a feasible solution? Is it possible to provide inheritance of some positive problem features under these conditions of structure permutations by the reordering procedure? In other words, is the genetic procedure under these specific conditions of the ALB problem more profitable than a simple random search?
2. What is the effectiveness of the exchange procedure, which performs a local optimization, in improving the GA results?
3. Does a choice of a specific crossover procedure affect the quality of results? Which crossover procedure, out of a wide set of crossover methods, is the most effective for the algorithm?
4. How the main parameters of the ALB problem (F-ratio – flexibility ratio of the task precedence constraints, WEST ratio – the ratio between number of elements and number of stations Na/Nst) affect the quality of GA solution?
5. How GA compares with MUST?
6. What is the effect of applying mutation in the GA on quality of solutions?

4.1. Comparative efficiency of GA

To answer the first and second questions, the GA was compared with a pure random version RAN in which the crossover operation was replaced by a procedure of random generation of new solutions, and with a version GA$^{\sim}$ in which the exchange procedure was excluded. These results are summarized in Fig. 3. Three methods were compared in this test: the complete implementation of the

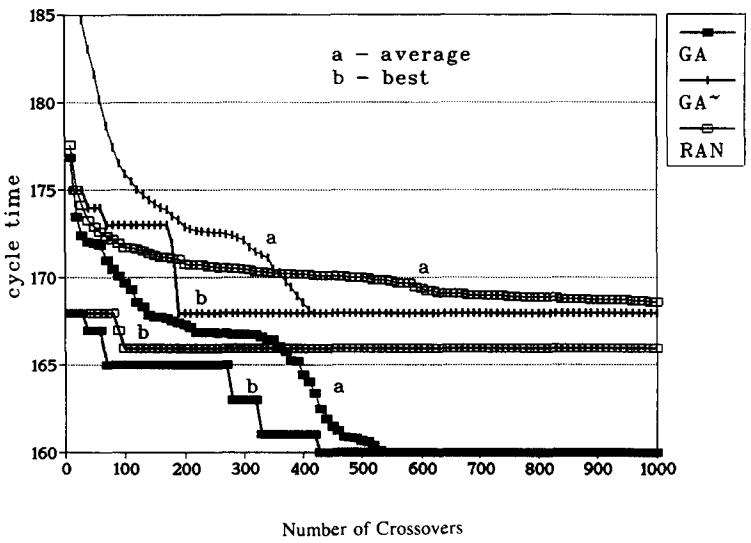

Fig. 3. Comparison of GA with a pure random version RAN and with a version GA$^{\sim}$ in which the exchange procedure was excluded.

genetic algorithm, GA, which includes the local optimization exchange procedure; a version of the genetic algorithm in which the exchange procedure was excluded, GA~; and a pure random version RAN. Solution quality is measured by the cycle time after each crossover/new generation creation. Two measures are shown in the graph: AVG – the average cycle time, and BST – the best cycle time for the current population. Fig. 3 presents the results for a problem with Na = 107, WEST-ratio = Na/Nst = 3.57, F-ratio = 0.1. One can see from the graph that both versions of the genetic algorithm converge rapidly to solution qualities better than the random version; thus it can be concluded that some useful properties are transferred from generation to generation in GA, providing better search performance than in random search. In addition, in GA the average cycle time converges quickly to the value of the best cycle time; Consequently, the GA yields a population of high-quality solutions, from which the assembly line designer is free to choose. In the random search, the average cycle time values do not converge to the best cycle time even after 1000 generations.

The comparison of GA with its version GA~ from which the exchange procedure was excluded, shows that local optimization through the exchange procedure allows to achieve much better results; In addition, the GA~ population of solutions converges to a relatively poor (high) cycle time value.

4.2. Crossover procedure selection

To select the most appropriate crossover operation 6 crossover methods were compared. Five of these methods were used in previous studies of genetic algorithms [8–12], while the sixth Fragment Reordering Crossover was developed in this work specifically for the ALB problem. In implementing these different crossover procedures, it was defined that all of them operate with two parent solutions (here – ordered vectors) P1 and P2 and produce two offspring (children). The first offspring (CH) is constructed as described below, and the second one is created in the same method, but with the roles of the parents exchanged. Some of the crossover procedures described below operate on a randomly defined fragment of the solution vector – a subset of adjacent elements between two randomly selected crossover sites.

Partially mapped crossover (PMX)

This operation was suggested by Goldberg and Lingle [8] and is aimed to maintain inheritance of adjacency and relative order of elements in the solution structure.

First, all elements from P1 are being copied to the same positions of CH; than using pairwise exchange some elements of CH are being reallocated in order to make a random fragment of CH to be an exact copy of the same fragment of P2.

Order crossover (ORD)

Developed by Davis [9], this operator preserves the order, adjacency and absolute positions of part of the elements and the relative order of the remaining elements.

CH inherits elements of a random fragment from P1, in the same order and position. The remaining elements are inherited from P2 in the order they appear in P2, beginning with the first position following the end of the random fragment and skipping over all elements already present in the CH.

Position based crossover (POS)

This procedure was proposed by Syswerda [10] and is intended to preserve inheritance of positional information.

A randomly chosen number of locations are selected at random in P1 and the elements in these positions are inherited by CH. The remaining elements are inherited in the order in which they appear in P2, skipping over all elements already included into CH. Although this operator is similar to order crossover (except the requirement of adjacency of elements being copied from P1), it was shown in [11] as well as in our study that it has really different properties.

Cycle crossover (CYC)

This crossover was suggested by Oliver et al. [12]. It allows to inherit the absolute positions of elements from parent structures.

The element of the start position of P1 (first or randomly defined) is being inherited by CH. The element which is in the same position in P2 cannot consequently be placed in this position. The position of this element is found in P1, and the element is inherited by CH to the same position. This continues until the cycle is completed by encountering the initial element from P1 in P2. All elements which were not yet copied to CH, are being copied from the same positions of P2.

Edge recombination crossover (EDG)

This procedure, which emphasizes adjacency information, was suggested by Starkweather et al. [11] for solving the Travelling Salesman Problem. It copies the start element from P1 to CH; then, beginning from this element, it tries to allocate an element to the next empty position using the following procedure:

- first it tries to find an element not inherited yet, which is adjacent to the element already allocated in the previous position in both P1 and P2;
- if there are not such elements, it tries to find an element which is adjacent to the element already allocated in the previous position either in P1 or in P2; and
- finally, if neither of these two types is found, an arbitrary not inherited yet element.

Fragment reordering crossover (FRG)

This procedure was developed in this work particularly for the ALB problem. All the other crossover operators, when applied on the ALB solution vectors, result in loss of feasibility of the offspring structures; consequently, to restore feasibility in regard to the precedence constraints, the reordering procedure is applied to offspring structures during the decoding (G4) step of GA. The FRG operator is aimed to maintain the inheritance of positions and the relative order of elements in the structure, providing changes within the fragment which do not violate the precedence constrains. This characteristic of preserving feasibility in regard of precedence constraints is unique to FRG. In all the other crossover procedures reordering is needed to restore a feasible solution; in this reordering process positive inheritance characteristics may become distorted.

The FRG procedure may be considered as a special case of POS and reversed version of ORD crossover. First, all elements from P1 are copied to the same positions of CH. Then all the elements of a random fragment in CH are reallocated within this fragment according to the order in which they appeared in P2. One can see that the precedence relations between any element within the fragment (regardless of its allocation in this fragment) and all elements outside the fragment in CH solution are inherited from P1. Precedence relations within the fragment are inherited from P2. Consequently, if P1 and P2 are feasible sequences, CH inherits this feasibility.

The experimental results show that CYC and FRG crossover procedures dominate all the rest. In Fig. 4 averages of the normed values of AVG for 12 problems (all given assembly networks for Nst = 10 and Nst = 30) are displayed for all crossover procedures. Whereas the poor performance of adjacency-oriented EDG are natural, the success of CYC is quite unexpected. It may be justified by possible importance of some absolute positions occupation by certain elements in conditions of precedence constraints.

To compare CYC and FRG more precisely the average values of three solution quality criteria were evaluated on the whole set of problems tested: BST – the shortest cycle time, AVG – the average cycle time of the population, and PS – the final population size defined by the number of unique solutions in the population after 1000 crossovers. PS is a measure of solutions diversity. In all the tests the initial population size was 50.

As it is shown in Fig. 5, FRG dominates CYC and provides more solutions with higher quality.

4.3. *Influence of ALB problem parameters*

Two parameters of the ALB problem: WEST-ratio and *F*-ratio were evaluated, while comparing the performance of GA with MUST. Whereas MUST aims to find all optimal (or near optimal) solutions, GA obtains a set of relatively good solutions, possibly with different cycle times. Although GA would probably perform best with very large ALB problems, having high values of *F*-ratio (measure of the strength of the precedence relations

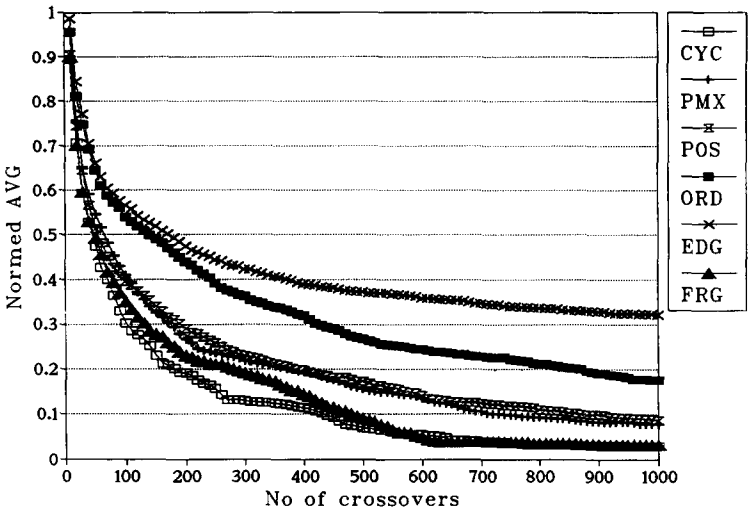

Fig. 4. Evaluation of crossover operators.

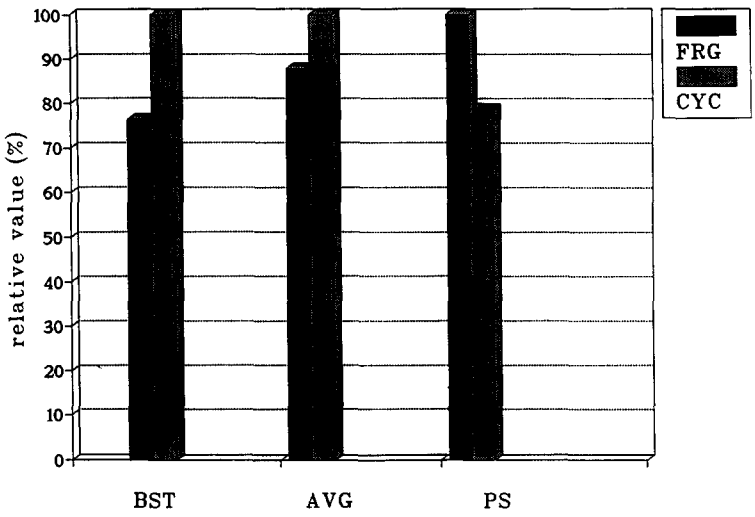

Fig. 5. Comparison of Cycle Crossover (CYC) and Fragment Reordering Crossover (FRG).

among activities [2]), the tests were performed on the entire problem population, including highly constrained problems with F-ratio of 0.1 for which the performance of MUST is excellent. The results of computational experiments show that WEST-ratio is the most important factor affecting the quality of GA solution. Fig. 6 contains average values of AVG increase above MUST solutions as a function of WEST-ratio. GA performance gets worse for small WEST-ratios (which may be explained by the limited size of solutions space) but AVG values still do not exceed MUST solutions by more than 4–5%. With a growth of WEST-ratio, the average value of cycle times from solutions provided by GA converges to the best optimal solutions of MUST very fast.

The dependance of AVG on the F-ratio is not so significant. The AVG improvement with F-ratio growth is very close to a linear function, as shown in Fig. 7.

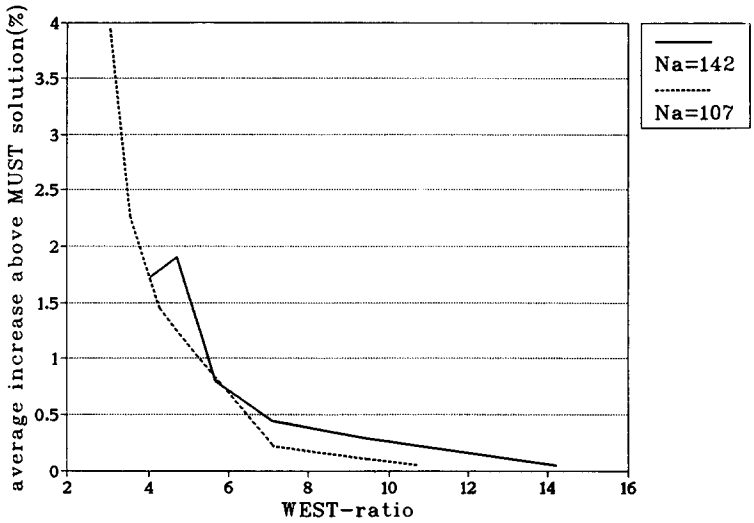

Fig. 6. Comparison of GA solutions to MUST as function of WEST-ratio.

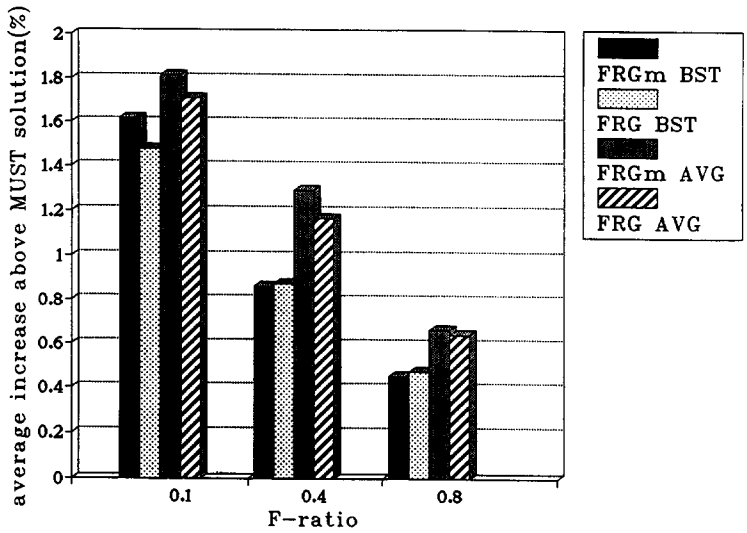

Fig. 7. Comparison of GA solutions to MUST as function of F-ratio.

4.4. *Mutation factor*

The mutation procedure in a GA is aimed to preserve or increase the diversity of solutions by creating new substructures of elements. The commonly used mutation procedure is based on exchange of elements between two randomly selected positions, which occurs in the genetic process with some predefined probability. Since in the GA algorithm a reordering procedure must be applied following the mutation to preserve precedence feasibility, it was assumed that the common mutation process will not provide a significant diversity of substructures even when applied with probability $p_m = 1$.

To increase the influence of mutation on the final offspring representation v, the following modification of a crossover operator was suggested (FRGm): with specified probability p_{mc} the offspring of usual FRG operator is being replaced

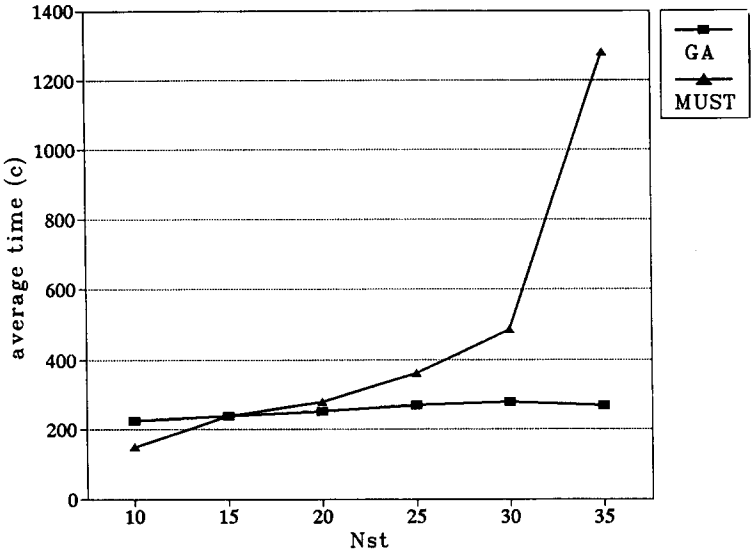

Fig. 8. Performance times of GA vs. MUST as function of Nst.

with randomly generated integer vector. Fig. 7 contains the results of comparison of BST and AVG values when FRG and FRGm were in use ($p_{mc} = 0.1$). The addition of FRGm operator causes improvement of BST in the zone of moderate and high flexibility, while AVG get worse. When precedence constraints are very strict, FRGm does not manage to improve BST values, and results only in slowing down the convergence of genetic process. So that use of mutations seems justified for ALB problems with high F-ratio.

4.5. *Computational effort*

GA and MUST run times were obtained on a Silicon Graphics workstation. The summary of these experimental results is given in Fig. 8. For each Nst, the average run times required for GA and MUST to solve 6 problems from the experimental set are presented. Whereas MUST effort strictly depends on Nst, GA is almost unsensitive to this parameter. One can see also that in term of computational efforts GA is much more effective than MUST when Nst > 20.

5. Conclusions

1. The genetic approach combined with a simple local optimization (element exchange) procedure shows promising results when applied for solving ALB problems. The suggested GA may be recommended for cases when solutions diversity is more important than their accuracy. The GA is much faster then MUST procedure for problems with large number of stations.

2. The crossover procedure choice is important part of GA-based applications development. The fragment reordering crossover (FRG), defined in this work, is the most appropriate method for ALB problems.

3. The GA reaches its best performance in the zone of high WEST-ratio and high problem flexibility (F-ratio) in which MUST, as an exhaustive enumeration procedure, faces computer resources limitations.

4. To increase diversity of solutions and improve the best solution in the population, additional mutations within FRGm procedure may be recommended, except for cases with extremely low assembly network flexibility.

Acknowledgement

This research was supported in part by the Fund for the Promotion of Research at the Technion.

References

[1] Mastor, A., 1970. An experimental investigation and comparative evaluation of production line balancing techniques. Mgmt. Sci., 16: 728–745.

[2] Dar-El (Mansoor), E.M., 1975. Solving large single-model assembly line balancing problems – A comparative study. AIIE Trans., 7: 302–310.

[3] Talbot, F.B., Patterson, J.H. and Gehrlein, W.V., 1986. A comparative evaluation of heuristic line balancing techniques. Mgmt. Sci., 32: 430–454.

[4] Dar-EL, E.M. and Rubinovitch, Y., 1979. MUST-a multiple solutions technique for balancing single model assembly lines. Mgmt. Sci., 25: 1105–1113.

[5] Goldberg, D., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, Reading, MA.

[6] Whitley, D. and Kauth, J., 1988. GENITOR: A different genetic algorithm. Proc. Rocky Mountain Conf. on Artifical Intelligence.

[7] Laszevski, G., 1991. Intelligent structural operators for the k-way graph partitioning problem. Proc. 4th Internat. Conf. of Genetic Algorithms and their Applications.

[8] Goldberg, D. and Lingle, R., 1985. Alleles, loci, and the Travelling salesman problem. Proc. Internat. Conf. of Genetic Algorithms and their Applications.

[9] Davis, L., 1985. Applying adaptive algorithms to epistatic domains. Proc. Internat. Joint Conf. on Artificial Intelligence.

[10] Syswerda, G., 1990. Schedule optimization using genetic algorithms, in: I. Davis (Ed.), Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York.

[11] Starkweather, T., McDaniel, S., Mathias, K., Whitly, D. and Whitly, C., 1991. A comparison of genetic sequencing operators. Proc. 4th Internat. Conf. of Genetic Algorithms and their Applications.

[12] Oliver, I., Smith, D. and Holland, J., 1987. A study of permutation crossover operations on the travelling salesman problem. Proc. 2nd Internat. Conf. of Genetic Algorithms and their Applications.