ARTICLE IN PRESS

Available online at www.sciencedirect.com

SCIENCE DIRECT®

European Journal of Operational Research xxx (2004) xxx–xxx

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

A genetic algorithm for robotic assembly line balancing

Gregory Levitin [a,*], Jacob Rubinovitz [b], Boris Shnits [b]

[a] *Division of Planning, Development and Technology, The Israel Electric Company Ltd., P.O. Box 10, Haifa 31000, Israel*
[b] *Faculty of Industrial Engineering and Management Technion, Israel Institute of Technology, Haifa 32000, Israel*

Abstract

Flexibility and automation in assembly lines can be achieved by the use of robots. The robotic assembly line balancing (RALB) problem is defined for robotic assembly line, where different robots may be assigned to the assembly tasks, and each robot needs different assembly times to perform a given task, because of its capabilities and specialization. The solution to the RALB problem includes an attempt for optimal assignment of robots to line stations and a balanced distribution of work between different stations. It aims at maximizing the production rate of the line. A genetic algorithm (GA) is used to find a solution to this problem. Two different procedures for adapting the GA to the RALB problem, by assigning robots with different capabilities to workstations are introduced: a recursive assignment procedure and a consecutive assignment procedure. The results of the GA are improved by a local optimization (hill climbing) work-piece exchange procedure. Tests conducted on a set of randomly generated problems, show that the Consecutive Assignment procedure achieves, in general, better solution quality (measured by average cycle time). Further tests are conducted to determine the best combination of parameters for the GA procedure. Comparison of the GA algorithm results with a truncated Branch and Bound algorithm for the RALB problem, demonstrates that the GA gives consistently better results.

Keywords: Genetic algorithms; Assembly lines; Non-identical Robots; Productivity; Hill climbing

1. Introduction

1.1. Problem description and previous work

An increasing requirement for flexibility of production is motivated by fast changes in technology and by customers demand for greater product variety. The main method of providing the desired flexibility is development of flexible assembly

* Corresponding author. +972 4 8183726; fax: +972 4 8183790.
E-mail addresses: levitin@iec.co.il (G. Levitin), ierjr01@ie.technion.ac.il (J. Rubinovitz), shnitsb@tx.technion.ac.il (B. Shnits).

0377-2217/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2004.07.030

systems (FAS), equipped with assembly robots (Owen, 1985). Robots play an important role in flexible assembly systems. One important configuration of robots in flexible assembly is the use of robotic assembly lines. The rationale for performing assembly with robots in an assembly line configuration is due to specialization in operations. Usually, specific tooling is developed to perform the activities needed at each station. Such tooling is attached to the robot at the station, in order to avoid the time waste required for tool change. The design of the tooling can take place only after the line has been balanced. Balancing of the robotic assembly lines includes two main objectives: to achieve an optimal balance on the assembly line for a given number of assembly cells (stations) or given required production rate, and to allocate the best fitting robot to each station. Different robot types may exist at the assembly facility. These robots need to be re-assigned when a new product is planned for assembly. Each such robot type may have different capabilities and performance times for various elements of the assembly task. Unlike manual assembly lines, where actual times for performance of activities vary considerably and optimal balance is rather of theoretical importance, the performance of robotic assembly lines depends strictly on the quality of its balance, and on robot assignment.

Graves and Holmes (1988) suggest an algorithm for assignment of activities and equipment to assembly line stations, satisfying the annual production rate. The objective of their work is to minimize total cost that is composed of fixed equipment and tooling costs, variable equipment usage and set-up costs. Their algorithm finds the minimum cost configuration for the mixed-product assembly line using a single assembly sequence for each product. Since most assembled products may be assembled using several alternative sequences, this algorithm finds only a local optimum, and does not take advantage of the assembly task flexibility. As a result, it cannot find a solution minimizing idle time at each station, whereas for robotic assembly lines, such optimal balancing is very important.

Rubinovitz and Bukchin (1991) were the first to formulate the robotic assembly line balancing problem (RALB) as one of allocating equal amounts of work to the stations on the line while assigning the most efficient robot type from the given set of available robots to each workstation. Their objective was to minimize the number of workstations for a given cycle time (productivity) of the line. They formulated the following assumptions:

1. The precedence relationship among assembly activities is known and invariable. This precedence is due to technological assembly constraints, and is represented by a precedence graph.
2. The duration of an activity is deterministic. Activities cannot be subdivided.
3. The duration of an activity depends on the assigned robot.
4. There are no limitations on assignment of an activity or a robot to any station other than the precedence constraints and the robots ability to perform the activity.
5. A single robot is assigned to each station.
6. Material handling, loading and unloading times, as well as set-up and tool changing times are negligible, or are included in the activity times. This assumption is realistic on a single-model assembly line, that works on the single product for which it is balanced. Tooling on such robotic line is usually designed such that tool changes are minimized within a station. If tool change or other type of set-up activity is necessary, it can be included in the activity time, since the transfer lot size on such line is of a single product.
7. All types of robots are available without limitations. The purchase cost of the robots is not considered.
8. The line is balanced for a single product.

The RALB algorithm (Rubinovitz and Bukchin, 1991; Rubinovitz et al., 1993) is based on a Frontier-Search modification of the Branch-and-Bound method. It builds a search tree by assigning robots and task elements to stations. As a lower bound, the sum of minimal possible times for activities not yet assigned to stations is used. To maintain the huge number of nodes on the search tree, the algorithm may require more storage space than available. It also requires significant computation time. As a result, the Branch-and-Bound based algorithm, even with heuristic rules incorporated to reduce the search space, can be used for solving relatively small problems. This approach has been generalized by Bukchin and Tzur (2000), to design a flexible assembly line when several equipment alternatives are available. The objective is to minimize equipment cost. An exact Branch and Bound algorithm is developed to solve moderate problems, in which a heuristic procedure is incorporated to cope with large problems.

Kim and Park (1995) focus on the problem of assigning assembly tasks, parts and tools on a serial robotic assembly line so that the total number of robot cells required is minimized while satisfying the various constraints. Assignment of robots with different performance capabilities is not part of their model. They suggest an integer programming formulation of this problem and a strong cutting plane algorithm to solve it.

Khouja et al. (2000) suggest statistical clustering procedures to design robotic assembly cells. The proposed methodology has two stages. In the first, a fuzzy clustering algorithm is employed to group similar tasks together so that they can be assigned to robots while maintaining a balanced cell and achieving a desired production cycle time. In the second stage, a Mahalanobis distance procedure is used to select robots appropriate for the task groups (for more details on the Mahalanobis metric and its applications for clustering, see Mahalanobis, 1936; Everitt, 1974). While their work focuses on a robotic cell design, it seems that the approach can be extended to design of a line of cells with similar cycle times. However, in an assembly line, task elements may be assigned to a single robot based on the robot capabilities, and not on task similarity, as assumed in their work.

Nicosia et al. (2002) deal with the problem of assigning operations on a production line to an ordered sequence of non-identical workstations, while observing precedence relationships and cycle time restrictions. The objective is to minimize the cost of the workstations. This formulation is very similar to the RALB problem. The approach used to solve the problem is by a dynamic programming algorithm with several fathoming rules used to reduce the number of states. The authors classify instances of the problem that are polynomially solvable.

1.2. Methodology and notation

This paper suggests an algorithm for solving large and complex RALB problems. This algorithm minimizes the cycle time of an assembly line with the given number of stations. It provides a solution on how to group N_a work activities performed at N_st stations and how to assign a single robot of one of N_r types to each station so as to achieve a minimal cycle time (the maximum time required for assembly at any given station). The algorithm is based on the genetic approach, which uses a simple principle of evolution. Combinatorial explosion of the storage requirements does not occur with the increase of the problem size as in the Branch-and-Bound method.

Only simple procedures are needed in GA for the estimation of solution quality. These may be easily changed or modified, providing a desirable flexibility of tools for real robotic assembly lines.

Notation:

N_st	total number of stations
N_a	total number of activities
N_r	total number of different types of robots
P	precedence matrix in which each element p_{ij} is 1 if activity i immediately precedes activity j and 0 otherwise
Y_i	set of immediate predecessors of activity i
$t_{r,j}$	time of performance of jth activity by robot r (if activity j can not be performed by the robot r, $t_{r,j} = \infty$)
τ_j	average performance time for activity j
$r(s)$	number of robot assigned to station s
$s(j)$	number of station to which activity j is assigned
T_s	total execution time for station s
C_0	initial estimation of assembly line cycle time
v	integer vector of numbers of activities representing feasible solution.

In Section 2 of this paper, the adaptation of the genetic algorithm for RALB problem is described. Section 3 presents the result of testing the performance evaluation of the algorithm for the different procedures suggested. Conclusions are presented in Section 4.

2. The genetic algorithm for RALB

The comprehensive description of GAs theory can be found in Goldberg (1989). A bibliography of numerous applications of GA in manufacturing is available in Alander (1995). Falkenauer (1998) provides an in-depth discussion of industrial applications of grouping GAs. In Rubinovitz and Levitin (1995) the application of GA for the simple, single-model, assembly line balancing (SALB) is described and reasons for choosing this approach are discussed. One of the advantages of GAs for the SALB problem is the ease of handling different evaluation functions. As a result, this approach has been further explored by other researchers, mainly to cope with the multiple objectives of an assembly line (Kim et al., 1996; Mitsuo Gen et al., 1996; Suresh et al., 1996; Kim et al., 2000; Ponnambalam et al., 2000; Sabuncuoglu et al., 2000).

This paper presents modification of the method suggested by Rubinovitz and Levitin (1995) for the more complicated RALB problem, which involves the selection and assignment of robots with different performance capabilities to workstations.

Unlike various constructive optimization algorithms that use sophisticated methods to obtain a single good solution, the GA deals with a set of solutions (population) and tends to manipulate each solution in the simplest way. "Chromosomal" representation requires the solution to be coded as a finite length string. The basic steps of GENITOR version of GA (Whitley, 1989), used in this paper, are as follows:

G1. Generate an initial population of randomly constructed chromosomes (structures) that represent solutions of the problem. Evaluate the fitness of each solution (see step G4).

G2. Select at random two solutions and produce a new solution (offspring) using a crossover procedure that provides inheritance of some basic properties of the parent structures in the offspring. (Some genetic algorithm schemes suggest a selection with a bias proportional to the solution quality; this is not the case here.)

G3. Allow the offspring to mutate with mutation index p_m, which results in slight changes in the offspring structure and maintains diversity of solutions. This procedure avoids premature convergence to a local optimum and facilitates jumps in the solution space.

G4. Decode offspring to obtain the objective function (fitness) values. These values are a measure of quality that is used to compare different solutions.

G5. Apply a selection procedure that compares new offspring with the worst solution in the population. The better solution joins the population and the worse one is discarded (removed from the population). If the population contains equivalent structures following selection, redundancies are eliminated and, as a result, the population size decreases slightly.

G6. Terminate the algorithm if after repeating steps G2–G5 Z times no improvement of the best-in-population solution was achieved (Z is a preliminarily specified parameter).

A classical permutation encoding is used to create a genotype, and a procedure is applied to transform each genotype permutation into a feasible problem solution before evaluating it (as described in detail in Rubinovitz and Levitin, 1995). Therefore the fitness evaluation procedure (step G4) is the only one that is tightly connected with the nature of the problem being solved. This step must include transformation operators (if it is necessary) and procedures for quality criteria evaluation. Inclusion of some local optimization procedures into the transformation procedure can also significantly improve performances of GA. Some optimization methods may also be implemented in the stage of initial population generation (step G1). The implementation of these

GA elements in adapting a GA algorithm for the RALB problem is described in the following sections.

2.1. Solution representation and the basic GA procedures

The choice of solution representation (structure) affects the method of transformation and evaluation. In this work we use a representation of RALB problem solution that includes three integer vectors:

1. Vector v containing a permutation of task element (activity) numbers, ordered according to their technological precedence sequence.
2. Vector of pointers to the position of the first activity for each station. These pointers divide the vector of activities into N_{st} parts.
3. Vector of robot numbers (indicating robot types) assigned to each station.

This solution representation scheme for a sample problem is presented in Fig. 1.

Only the first vector (ordered sequence of activities) is involved in the genetic process (crossover, mutation and selection procedures) in our algorithm. The two other vectors are generated, for each solution represented by the first vector and produced by the GA, by a set of simple decoding procedures.

It should be noted that such representation allows equal solutions to be represented by different vectors (because of feasible activities permutations within stations). Thus an appropriate procedure is necessary in order to check the identity of solutions.

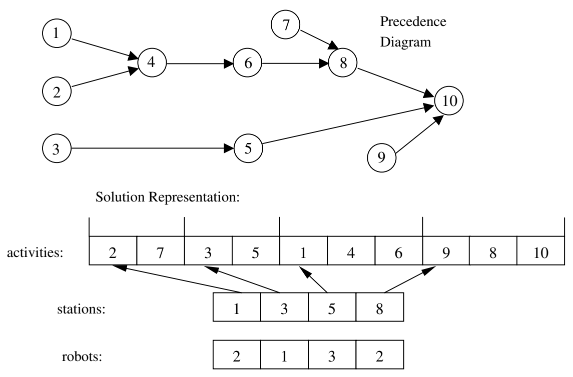

Fig. 1. Solution representation scheme for a sample problem.

2.2. Crossover and mutation operators

For a given solution representation, we can now define crossover and mutation operations. Most of the crossover procedures suggested by Starkweather et al. (1991) operate with two parent solutions, and produce two offspring (children). In this work, the Fragment Reordering Crossover which was introduced by Rubinovitz and Levitin (1995), is used. This crossover procedure preserves solutions feasibility in problems with precedence constraints. The Fragment Reordering Crossover works as follows:

- All elements from the first parent are copied to identical positions in the offspring string.
- A fragment of the offspring string is defined as a subset of adjacent elements between two randomly selected positions (crossover sites).
- All the elements within the fragment are reordered according to the order of their appearance in the second parent vector.

The second offspring is generated in the same method, with the roles of its parents reversed.

In the following example the elements of a randomly chosen fragment in the first parent **P1** are marked with bold font as well as corresponding elements in the second parent **P2**. **O** is an offspring solution obtained by Fragment Reordering Crossover.

P1:	1	2	**3**	**4**	**5**	**6**	**7**	8	9	10
P2:	**7**	8	9	2	**4**	**5**	1	**3**	**6**	10
O:	1	2	**7**	**4**	**5**	**3**	**6**	8	9	10

The mutation procedure selects two positions within the solution string at random, and looks for a pair of elements closest to these two positions that can swap places without violating the precedence constraints (i.e. preserving solution feasibility). The elements found in this way swap positions in the solution string.

2.3. Decoding procedures

The purpose of the set of decoding procedures is to provide the following functions:

1. Transformation of an arbitrary sequence of activities into a feasible one (this procedure needs to be performed only for the initial randomly generated solutions, because the crossover and mutation operators used further on preserve feasibility of solutions).
2. Partition of the sequence i.e. assignment of activities to the N_{st} stations.
3. Assignment of robots to stations.
4. Evaluation of the cycle time for the given balance.
5. Local improvement of the solution (if possible).

The transformation of an arbitrary sequence of activities into a feasible one (step 1) is performed by a re-ordering procedure that restores feasibility of a randomly generated string according to the precedence constraints. Detailed description of this procedure can be found in Rubinovitz and Levitin (1995). For sake of clarity, this procedure is illustrated for a sample problem in Fig. 2, and explained below.

The feasible vector in Fig. 2 is based on problem precedence diagram as presented in Fig. 1. The re-ordering procedure that generates this vector from a feasible vector is summarized below:

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.

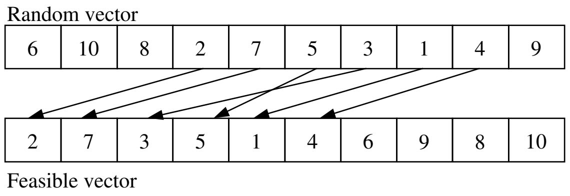

Fig. 2. A re-ordering procedure that restores feasibility of a randomly generated string according to the precedence constraints of the sample problem of Fig. 1.

Let us introduce a logical function $W_i(j)$ which returns true if activity i can be moved from station $s(i)$ to station j and false otherwise:

$W_i(j) =$ false if

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

or

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

$W_i(j) =$ true otherwise.

The re-ordering procedure, as illustrated in Fig. 2, consists of the following steps:

```
R1. For all activities $1 \leqslant i \leqslant N_a$ assign $s(i) = 2; k = 1$.
R2. Find the least $m$ $(m \leqslant N_a)$:
    $s(v\sim(m)) = 2$ and $W_{v\sim(m)}(1) =$ true.
R3. If such $m$ does not exist end of procedure.
    Else: $v(k) \leftarrow v\sim(m)$;
        $k \leftarrow k+1$;
        $s(v\sim(m)) \leftarrow 1$;
        return to R2.
```

Two alternative procedures were developed for assignment of activities and robots to different stations (steps 2–3): a recursive procedure and a successive assignment procedure. These procedures, developed for the RALB problem, are discussed and illustrated in detail in the following sections.

The local improvement function (step 5) was performed by an exchange procedure, like the one used in Rubinovitz and Levitin (1995).

2.3.1. Recursive assignment procedure (*R*)

This procedure aims to assign activities to stations without violating the v sequence. The recursive procedure was developed in order to divide a vector v into $M = N_{st}$ parts, while trying to achieve the maximal equality of total execution times for all stations.

First, it defines the average performance time for each activity i as:

$$\tau_i = \sum_{r=1}^{N_r} t_{r,i}\delta_{r,i} \Bigg/ \sum_{r=1}^{N_r} \delta_{r,i} \tag{1}$$

where $\delta_{r,i} = 0$ if $t_{r,i} = \infty$, and $\delta_{r,i} = 1$ otherwise.

Next, the procedure divides the total vector v (i.e. the set of its elements from the left position $\mathrm{pl} = 1$ to the right position $\mathrm{pr} = N_a$) into two parts with ratio H/Q where $H = [M/2]$ and $Q = M - H$. To do this it finds a position $i(\mathrm{pl} \leqslant i \leqslant \mathrm{pr})$ such that a time ratio value TR:

$$\mathrm{TR} = \sum_{j=\mathrm{pl}}^{i} \tau_{v(j)} \Bigg/ \sum_{j=i+1}^{\mathrm{pr}} \tau_{v(j)} \tag{2}$$

is as close as possible to the ratio H/Q. Such i should minimize the imbalance function $d(i)$:

$$d(i) = \left| Q \sum_{j=\mathrm{pl}}^{i} \tau_{v(j)} - H \sum_{j=i+1}^{\mathrm{pr}} \tau_{v(j)} \right| \tag{3}$$

Using Eq. (2), the procedure finds a value for i that divides the initial vector into two subvectors ($\mathrm{pl} = 1$; $\mathrm{pr} = i$ and $\mathrm{pl} = i + 1$; $\mathrm{pr} = N_a$). These resulting vectors must be further divided into $M = H$ and $M = Q$ parts respectively using the same procedure recursively until $M = 1$. At the end of the recursion, the total execution time is calculated and boundary positions pl and pr for all stations are fixed.

Having all activities assigned to the stations, the procedure chooses robots to minimize the total execution time for each station:

$$r(s) = \arg_{1\leqslant h\leqslant N_r} \left\{ T_s(h) = \sum_{k=\mathrm{pl}_s}^{\mathrm{pr}_s} t_{h,v(k)} = \min \right\} \tag{4}$$

where pl_s and pr_s are the first and the last elements of a fragment of the vector v corresponding to station s.

For a given station s minimal T_s may be equal to ∞. This means that no single robot can perform the activities assigned to this station. The GA discards such a solution.

An example of the procedure is presented in Fig. 3a. The performance times for the example are presented in Table 1.

2.3.2. Consecutive assignment procedure (C)

Similarly to the recursive procedure, the consecutive procedure divides the vector v into N_{st} parts, thus distributing activities given in a defined sequence among stations, and assigning robots to the stations. For a given initial value of cycle time C_0, the procedure attempts to allocate activities to a station using robots that allow to maximize the number of activities performed at each station. For robots that result in the same number of activities, the procedure will choose a robot that minimizes the total execution time of the station.

For each station s (where $1 \leqslant s \leqslant N_{\mathrm{st}}$), it defines the set of preferred robots Ω_s as follows:

$$k \in \Omega_s \text{ if } m(k) \geqslant m(h) \quad \text{for } 1 \leqslant h \leqslant N_r \tag{5}$$

where $m(h)$ is the maximal number of activities robot h can perform in the given sequence (vector v) during a time not greater than C_0:

$$T_s(h) = \sum_{k=\mathrm{pl}_s}^{\mathrm{pl}_s+m(h)} t_{h,v(k)} < C_0 \leqslant \sum_{k=\mathrm{pl}_s}^{\mathrm{pl}_s+m(h)+1} t_{h,v(k)} \tag{6}$$

Next, it defines the robot to be assigned to the sth station as

$$r(s) = k \text{ if } T_s(k) \leqslant T_s(h) \quad \forall h \in \Omega_s \tag{7}$$

and calculates the start position for the next station:

$$\mathrm{pl}_{s+1} = \mathrm{pr}_s + 1 = \mathrm{pl}_s + m(r(s)) + 1 \tag{8}$$

Beginning with initial value C_0, the procedure runs repeatedly while it fails to find any cycle time feasible allocation of activities (some activities remain unallocated). Before each new pass the value of C_0 is incremented by one. The procedure stops when a feasible allocation is achieved. The initial value of C_0 is determined as lower bound estimation of the system cycle time:

$$C_0 = \left\lceil \sum_{j=1}^{\mathrm{Na}} \min_{1\leqslant i\leqslant N_r} t_{i,j} \Big/ N_{\mathrm{st}} \right\rceil \tag{9}$$

An example of the procedure outcome for performance times from Table 1 is presented in Fig. 3b. This example, as the one used to illustrate the recursive procedure, is also solved for $N_{\mathrm{st}} = 4$. Using Eq. (9) to calculate initial (lower bound) estimate for C_0 we get: $C_0 = \lceil (12 + 30 + 19 + 23 + 27 + 10 + 14 + 19 + 12 + 17)/4 \rceil = \lceil 183/4 \rceil = 46$. It is not possible to find a solution for four stations within this cycle time. As a result, C_0 is incremented by one, but the procedure fails to find a solution for four stations until

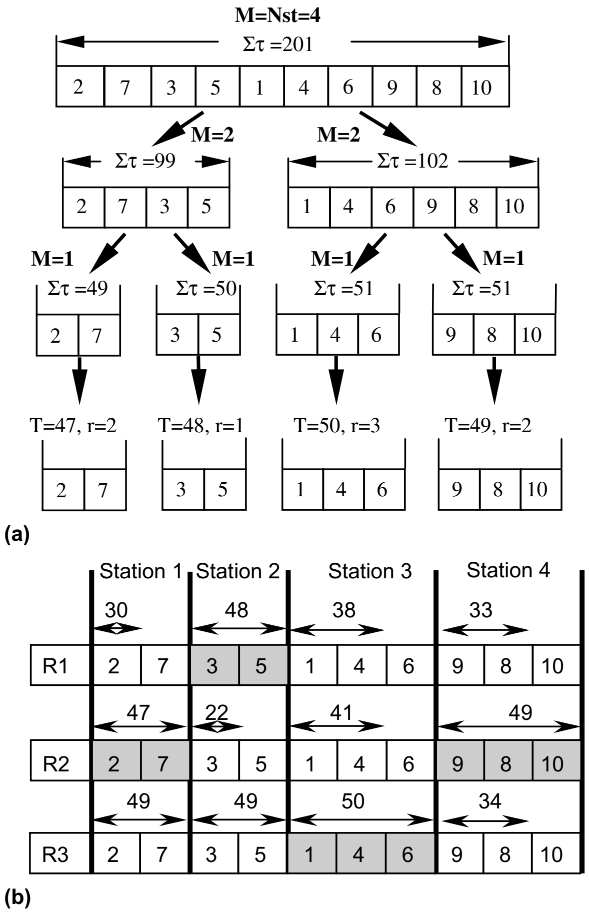

Fig. 3. (a) Example of the recursive assignment procedure. (b) Example of the consecutive assignment procedure.

Table 1
Performance times for 10 activities

Activity	1	2	3	4	5	6	7	8	9	10
$t_{1,j}$	12	30	19	26	29	14	21	19	14	20
$t_{2,j}$	15	33	22	26	31	10	14	20	12	17
$t_{3,j}$	15	30	22	23	27	12	19	21	13	17
τ_j	14	31	21	25	29	12	18	20	13	18

the value of $C_0 = 50$ is reached. At this cycle time, the solution presented in Fig. 3b is reached, with the robots assigned to each station according to Eqs. (5)–(7) highlighted in grey. In the solution illustration, the rows R1–R3 correspond to the three robot types available, and the row of the selected robot, highlighted in grey, indicates that this robot allows to maximize the number of activities performed at that station, while also minimizing the total execution time of the station. The double arrows indicate activities that can be performed at the station, by each robot, within the cycle time.

2.3.3. Exchange procedure (E)

First we introduce the logical function $W_i(q)$ which returns false if activity i cannot be transferred from station $s(i)$ to station q, and true otherwise:

$$\begin{aligned}&W_i(q) = \text{false if}\\&s(i) > q \text{ and } \exists k \in Y_i, \quad s(k) > q\\&\text{or}\\&s(i) < q \text{ and } \exists k : i \in Y_k, \quad s(k) < q\\&W_i(q) = \text{true otherwise.}\end{aligned} \tag{10}$$

Now consider two stations f and q with total execution times T_f and T_q ($T_f > T_q$). If exchange of activities i ($s(i) = f$) and j ($s(j) = q$) is feasible, the new execution times after the exchange are:

$$T^*_f = T_f - t_{r(f),i} + t_{r(f),j} \tag{11}$$

$$T^*_q = T_q - t_{r(q),j} + t_{r(q),i} \tag{12}$$

The exchange is worth-while if:

$$\max\{T^*_f, T^*_q\} < T_f \tag{13}$$

From these expressions one can derive the condition of exchange:

$$t_{r(f),j} < t_{r(f),i} \text{ and } T_q - T_f < t_{r(q),j} - t_{r(q),i} \tag{14}$$

The exchange procedure is as follows:

1. Rank all stations in order of total execution times.
2. For the most loaded station f and the other stations $1 \leqslant q \leqslant N_{\text{st}}$, $q \neq f$, in sequence (beginning from the least loaded), look for a pair of activities (i,j): $s(i) = f$, $s(j) = q$ which satisfies the conditions:

$$W_i(q) = W_j(f) = \text{true}, \tag{15}$$

$$p_{i,j} = p_{j,i} = 0. \tag{16}$$

3. If these conditions are satisfied as well as condition (14), perform the exchange, recalculate execution times T_f and T_q and return to step 1. If the desired pair of activities does not exist for all possible q, i and j, terminate the procedure.

2.3.4. Stop condition

The GA stops after performing a pre-defined number of cycles, NCYC, that is defined as a parameter. At a termination of each cycle, a "cataclysm" is performed, i.e. a new population of solutions is created, preserving only the best solutions. This is the usual procedure in GA to avoid convergence to local optimum. For each cycle, a pre-defined number of crossovers (NCRS) is performed, unless all solutions converge earlier to a single value of cycle time, without further improvement.

3. Performance evaluation

3.1. Evaluation of the assignment procedures

The assignment procedures were evaluated by conducting tests of the GA with each procedure for a large set of RALB problems with different characteristics are as follows.

F-ratio—Flexibility ratio, as defined by Dar-El (1973) measures the flexibility of the assembly task precedence constraints, by a ratio of the number of 0 elements in the precedence matrix (no precedence required) to the number of 1 elements in the matrix (hence tasks with no precedence required have an F-ratio of 1). Problems with three levels of F-ratio were generated and evaluated: low flexibility F-ratio = 0.1, medium flexibility F-ratio = 0.4, and high flexibility F-ratio = 0.8.

WEST ratio—work element to station number ratio, as defined by Dar-El (1973) measures the average number of activities per station. This measure indicates the expected quality of achievable solutions and the complexity of the problem. Problems with six levels of WEST ratios: 2, 3.33, 5, 7.5, 10 and 15 were generated and evaluated. These ratios were achieved by different combinations of problems with 20, 100 and 150 activities that were balanced for 10, 20 and 30 stations.

N_r—number of different robot types. This parameter affects problem complexity. Two levels were evaluated, with three and six robot types.

RF—robot (equipment) flexibility, as defined by Rubinovitz and Bukchin (1991) measures the number of different robot types that are capable to perform each activity. When RF = 0, each activity can be performed by a single robot type. When RF = 1, each activity can be performed by all robot types. The levels used in evaluation were 0, 0.33, 0.66, and 1.

RETV—robot expected time variability, measures the variability in activity performance times by different robot types. Three levels were tested, 0.1, 0.5 and 0.9 for low, medium and high variability.

The problems generated for this set of parameters were solved using each of the two assignment procedures with ten replications. The observed decision variables were the solution cycle time, and the time required to reach a solution. Results of the tests for low (0.1) and high (0.9) values of RETV are summarized in Figs. 4–7.

Analysis of test results shows that, in general, the Consecutive Assignment algorithm achieves better solution quality (measured by average cycle time). For all the parameter values tested, the Consecutive Assignment Procedure results in cycle times that are shorter or equal to those achieved with the Recursive Procedure. While solution quality tends to be similar for both procedures for low (0.1) RETV values, it is consistently better for high RETV values (both for RETV 0.5 (not shown), and for RETV 0.9) and for lower F-ratio

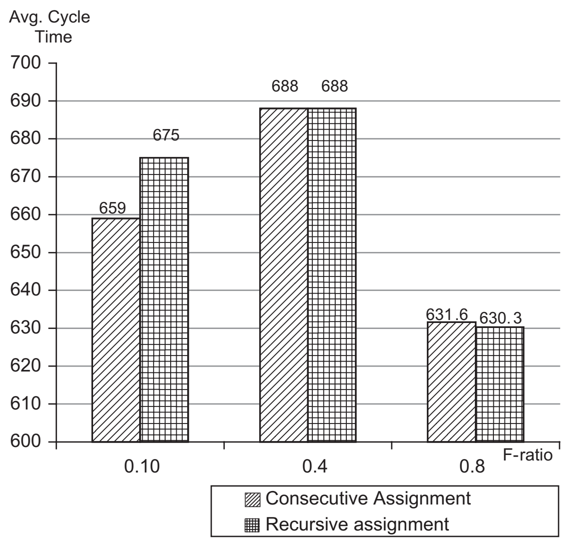

Fig. 4. Average cycle times vs. F-ratio levels for RETV = 10%.

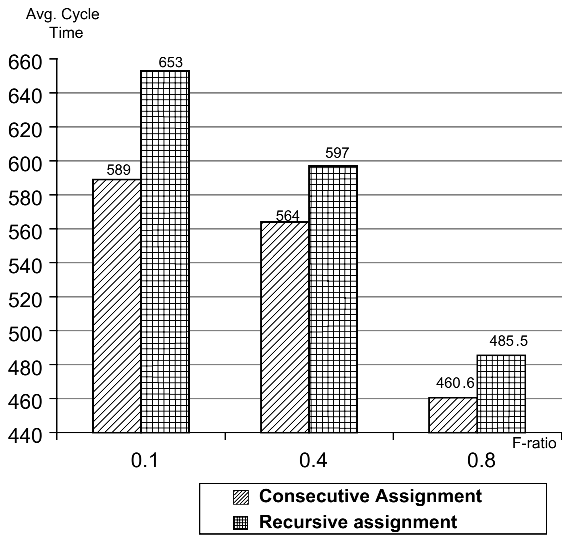

Fig. 5. Average cycle times vs. F-ratio levels for RETV = 90%.

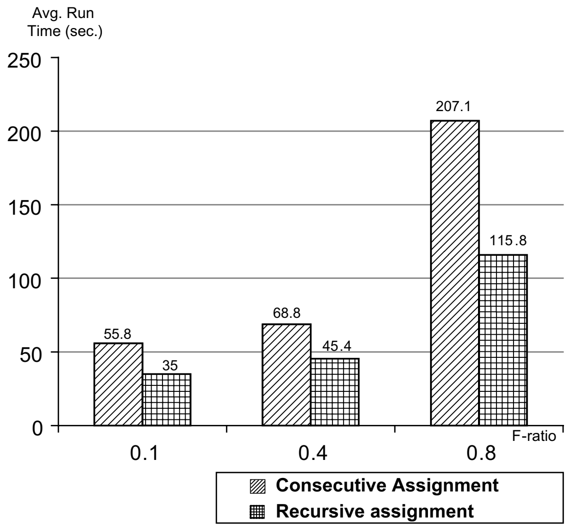

Fig. 6. Average run times vs. F-ratio levels for RETV = 10%.

values. The Recursive Assignment algorithm requires shorter run times when RETV values are very low, however the times required for both algorithms are shorter than single-digit number of minutes. However, for medium and high RETV values, and for problems with high F-ratio values, the Consecutive Assignment algorithm solves the problems in a significantly shorter time. In sum-

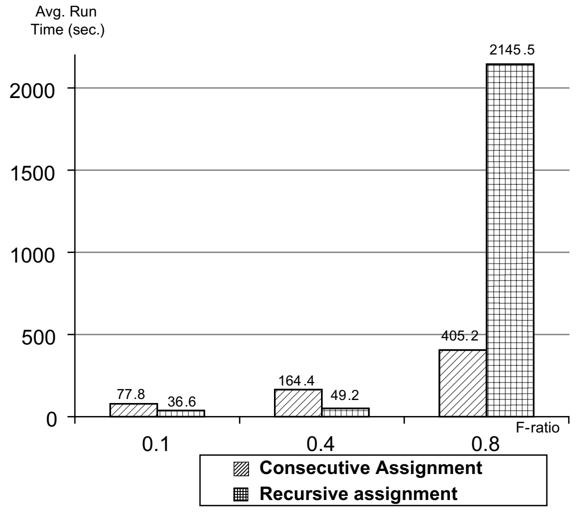

Fig. 7. Average run times vs. F-ratio levels for RETV = 90%.

mary, it is always recommended to use the Consecutive Assignment algorithm to solve the RALB problem, as the run time required is longer only for a small subset of the problems, and the solution quality is consistently superior.

The results above can be explained further, by analyzing the differences between the two procedures. The Recursive Procedure assigns activities to stations using activity times calculated as an average of the performance times with different robots. This means that for low time variability between the robots (low RETV values) this procedure will be more accurate. This explains its better performance for low RETV values, which is comparable to the performance of the Consecutive Assignment Procedure. Similar performance is also achieved by both procedures for high F-ratio values. This also can be explained by the larger solution space, that allows the approximate Recursive Procedure to find good solutions.

3.2. Selecting values for the GA parameters

The performance of a genetic algorithm, and the quality of solutions, can be affected by the values assigned to the different parameters of the algorithm. Extensive testing and tuning of the parameter values to be used with the consecutive assignment procedure of the GA was performed. Parameters that were evaluated in this set of tests are:

IPS—initial population size (the population size may decreases slightly, if equivalent structures are created during the selection process, in which case redundancies are eliminated).

NCYC—number of cycles, i.e. the number of times a new set of randomly generated solutions replaces the existing set, keeping only the best solutions in the population.

NCRS—number of crossovers performed in a single genetic algorithm cycle.

p_m—index of mutation, i.e. a random change by exchanging element positions in a solution string. p_m values between 0 and 1 indicate the probability that a newly generated solution string will undergo a single exchange of element positions. Integer p_m values greater than 1 indicate that every newly generated solution string will undergo p_m exchanges of element positions.

Different combinations of GA parameter values were tested for a representative set of eight RALB problems with different characteristics. The different parameter values tested are summarized in Table 2.

The eight different RALB problems were solved for all the combinations of the parameter values, with ten replications. The observed decision variables for each of the test runs were a solution cycle time (measure of solution quality), and a computer run time required (measure of algorithm performance). In selecting the recommended set of GA parameter values, preference was given to solution quality, as measured by the cycle time, over the performance, measured by computer run-times. This is due to a fact that most of the problems were solved within a single-digit number of minutes (on a personal computer) and there were no marked differences between them in terms of per-

Table 2
GA parameter values tested

Parameter	Values			
IPS	50		80	100
NCYC	5	20	50	80
NCRS	1500		3000	4500
p_m	0.5		1	2

formance times. To establish the recommended values for the GA parameters, the following procedure was used:

(1) Each of the eight different RALB problems was solved with all the 108 possible combinations of parameter values, and the combination resulting with the best solution quality (minimal cycle time) was found.
(2) For all other parameter combinations, the percentage of cycle time above the minimum found was calculated.
(3) For each combination of the parameter values, the average percentage of cycle time above the minimum cycle time value was calculated, based on the results of ten replications for the eight different RALB problems solved.
(4) For each distinct value used for each one of the parameters, the average percentage of cycle time above the minimum cycle time value was calculated. This was calculated based on an average of cycle time results for all the parameter combinations in which this particular parameter value was used.

The results of the average percentage of cycle time above the minimum cycle time, for the different parameter values tested, are summarized in Figs. 8–11.

It is evident from the test results that there were no marked differences in quality of solutions for all the problems, as all solutions were within 0.5 percent above the minimum cycle time. However,

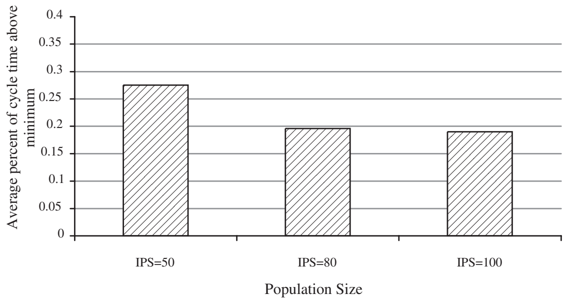

Fig. 8. Average percent of cycle time above the minimum cycle time, as function of the initial population size (IPS).

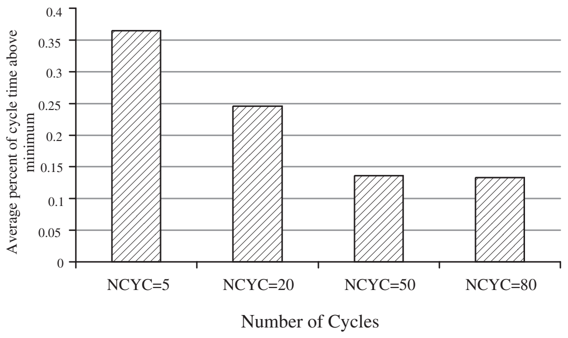

Fig. 9. Average percent of cycle time above the minimum cycle time, as function of the number of cycles (NCYC).

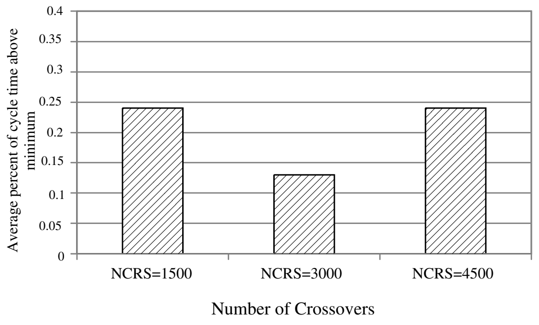

Fig. 10. Average percent of cycle time above the minimum cycle time, as function of the number of crossovers (NCRS).

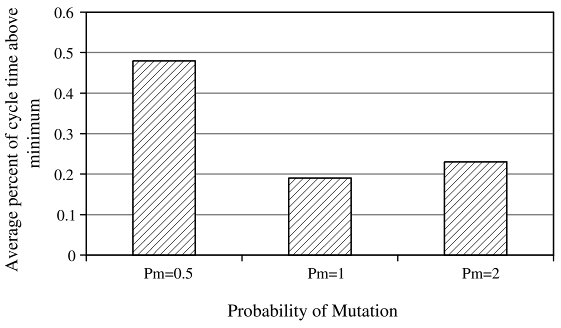

Fig. 11. Average percent of cycle time above the minimum cycle time, as function of the index for mutation (p_m).

based on the results, it is possible to recommend the values of IPS = 100, NCYC = 50, NCRS = 3000, and $p_m = 1$ for the GA parameters.

3.3. Comparison with a branch and bound algorithm

A reduced set of problems, with only three different robot types ($N_r = 3$), was used to compare the performance of the GA with the consecutive assignment procedure with the performance of the Branch and Bound algorithm for RALB suggested by Rubinovitz and Bukchin (1991). There is some difficulty in comparing the two algorithms: the GA minimizes the cycle time for a given number of stations, while the B&B algorithm has been developed to minimize the number of stations for a given cycle time. In order to make a performance comparison without changing either of the two algorithms, a set of problems with different problem parameters was generated, and a special comparison procedure was used. The problem characteristics used to generate this set of problems are summarized in Table 3. This experimental design consists of a total of 108 problems, for all the combinations of the parameters set. The comparison procedure was as follows:

- The GA with the consecutive assignment procedure was used to solve the entire set of problems, with ten replications (10 different problems were generated for each combination of the parameters set). An average cycle time for each parameter combination, resulting from the 10 replications, was calculated.
- The B&B algorithm was solved for the same set of problems, attempting to minimize the number of stations for the average cycle time achieved by the GA.
- A comparison was made between the number of stations used for the GA and the number of stations found by the B&B algorithm, for the same problems.

Table 3
Characteristics of the problem set

Parameter	Values			
F-ratio	0.1	0.4	0.8	
WEST ratio	2	5	10	
N_r		3		
RF	0	0.33	0.66	1
RETV	0.1	0.5	0.9	

B&B could solve only a small subset of 24 problems, of all the problems generated, to optimality. For this subset, equal quality solutions were achieved for 23 of the problems by both algorithms, and the B&B solution was better for only one problem. For all the other problems, the heuristic version of B&B had to be used. Both algorithms gave solutions of equal quality for about 30% of the problems. For the other problems the solutions achieved by the GA were of higher quality, balancing the line with one or two less stations. This superiority of solutions of the GA was demonstrated in particular for problems with greater complexity (high values of F-ratio and WEST-ratio). An important advantage of the GA is that the time required to reach a solution is not growing with problem complexity, and is dependent on the set of parameters of the GA (initial population size, number of cycles and number of crossovers). Most solutions were achieved in few seconds (none exceeded a single-digit number of minutes) on a Pentium3 personal computer.

3.4. Consistency and robustness of the Genetic Algorithm

In order to check the consistency and robustness of the GA, five representative problems with very different problem characteristics were selected. Each problem was solved by the GA ten times, using different problem parameters (such as initial population size), and randomly generating different initial populations. At selected intervals in the solution process (after a given number, x, of crossovers), the best solution values were recorded. The purpose was to show that the different solutions of a problem converge to a common value of the solution's quality. The basic idea behind this test was that if we can show convergence to similar solution quality, even when changing parameters such as the initial population size, then we can conclude that this is not a random event, and the algorithm is consistent and robust. This was measured at each interval by calculating the coefficient of variance between the best solution values of the 10 GA runs. In three of the five problems tested, the algorithm converged to solutions of equal quality (coefficient of

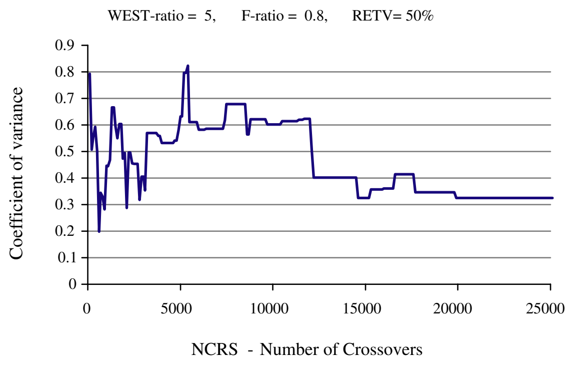

Fig. 12. Coefficient of variance between the best solutions of 10 different algorithm runs as function of number of crossovers (NCRS).

variance equal zero). In the two other problems of higher complexity (high F-ratio and WEST-ratio values) the algorithm converged to coefficients of variance of 0.11% and 0.31%, respectively. The convergence graph for the last problem tested is shown in Fig. 12. This graph shows that after about 20,000 crossovers, different parameters used for the GA all yield solutions of very similar quality (Coefficient of variance of less than 0.31%).

4. Conclusion

The objective of this work was to develop an efficient solution for the robotic assembly line balancing (RALB) problem. This solution aims to achieve a balanced distribution of work between different stations (balance the line) while assigning to each station the robot best fit for the activities assigned to it. The result of such solution would be an increased production rate of the line (by achieving a minimal cycle time).

Two different procedures for adapting the GA to the RALB problem and assigning robots to stations are introduced: a recursive and a consecutive procedure. Local exchange procedure is used to further improve the quality of solutions. The best combination of these procedures and GA parameters is reached by testing on an extensive set of randomly generated problems. The GA developed is shown to be consistent and robust. It achieves solutions of higher quality than a Branch and Bound algorithm, and solves large and complex problems very efficiently.

References

Alander, J., 1995. An indexed bibliography of genetic algorithms in manufacturing. In: Lance, C. (Ed.), Practical Handbook of Genetic Algorithms New Frontiers, vol. II. CRC Press, Boca Raton, FL.

Bukchin, J., Tzur, M., 2000. Design of flexible assembly line to minimize equipment cost. IIE Transactions 32, 585–598.

Dar-El (Mansoor), E.M., 1973. MALB—a heuristic technique for balancing large single-model assembly lines. AIIE Transactions 54, 343–356.

Everitt, B.S., 1974. Cluster Analysis, first ed. John Wiley & Sons, New York.

Falkenauer, E., 1998. Genetic Algorithms and Grouping Problems. John Wiley & Sons Inc.

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

Graves, S.C., Holmes, C.R., 1988. Equipment selection and task assignment for multiproduct assembly system design. The International Journal of Flexible Manufacturing Systems 1, 31–50.

Kim, H., Park, S., 1995. Strong cutting plane algorithm for the robotic assembly line balancing problem. International Journal of Production Research 33 (8), 2311–2323.

Kim, Y.K., Kim, Y.J., Kim, Y., 1996. Genetic algorithms for assembly line balancing with various objectives. Computers and Industrial Engineering 30 (3), 397–409.

Kim, Y.K., Kim, Y., Kim, Y.J., 2000. Two-sided assembly line balancing: a genetic algorithm approach. Production Planning and Control 11 (1), 44–53.

Khouja, M., Booth, D.E., Suh, M., Mahaney Jr., J.K., 2000. Statistical procedures for task assignment and robot selection in assembly cells. International Journal of Computer Integrated Manufacturing 13 (2), 95–106.

Mahalanobis, P.C., 1936. On the Generalized Distance in Statistics. National Institute of Science in India 12, 49–55.

Gen, M., Tsujimura, Y., Li, Y., 1996. Fuzzy assembly line balancing using genetic algorithms. Computers and Industrial Engineering 31 (4), 631–634.

Nicosia, G., Paccarelli, D., Pacifici, A., 2002. Optimally balancing assembly lines with different workstations. Discrete Applied Mathematics 118, 99–113.

Owen, A.E., 1985. Assembly with Robots. Kogan Page Ltd.

Ponnambalam, S.G., Aravindan, P., Naidu, G.M., 2000. A multi-objective genetic algorithm for solving assembly line balancing problem. International Journal of Advanced Manufacturing Technology 16, 341–352.

Rubinovitz, J., Bukchin, J., 1991. Design and balancing of robotic assembly lines. In: Proceedings of the Fourth World Conference on Robotics Research, Pittsburgh, PA.

Rubinovitz, J., Bukchin, J., Lenz, E., 1993. RALB—a heuristic algorithm for design and balancing of robotic assembly lines. CIRP Annals 42 (1), 497–500.

Rubinovitz, J., Levitin, G., 1995. Genetic algorithm for assembly line balancing. International Journal of Production Economics 41, 343–354.

Sabuncuoglu, I., Erel, E., Tanyer, M., 2000. Assembly line balancing using genetic algorithms. Journal of Intelligent Manufacturing 11, 295–310.

Starkweather, T., McDaniel, S., Mathias, K., Whitley, D., Whitley, C., 1991. A comparison of genetic sequencing operators. In: Proceedings of the Fourth International Conference of Genetic Algorithms and their Applications.

Suresh, G., Vinod, V.V., Sahu, S., 1996. A genetic algorithm for assembly line balancing. Production Planning and Control 7 (1), 38–46.

Whitley, D., 1989. The GENITOR algorithm and selective pressure: why rank-based allocation of reproductive trials is best. In: Proceedings of the Third International Conference on Genetic Algorithms and their Applications.