"Vasile Alecsandri" University of Bacău
Faculty of Sciences
Scientific Studies and Research
Series Mathematics and Informatics
Vol. 19 (2009), No. 2, 119 - 132

A HYBRID GENETIC ALGORITHM FOR BALANCING ASSEMBLY LINES WITH COMPATIBILITY CONSTRAINTS

OCTAV BRUDARU, DIANA POPOVICI and CINTIA COPACEANU

Abstract. The paper presents a hybrid genetic algorithm for deterministic assembly line balancing (ALB) problem with a single model and an additional constraint, which requires that the workstations are compatible with a given cover of the assembly tasks. The performance criteria are the minimizing of the idle time and the smoothing index. The algorithm includes a special procedure to generate the cover sets and a special mutation operator preserving the topological order. It is also combined with an efficient greedy procedure proper to the problem. All genetic operators are applied with dynamic probabilities that favour the creating and preserving of good constructive blocks. The experimental investigation proves the ability of the hybrid method to find good solutions to this type of balancing problem.

I. INTRODUCTION

The assembly line balancing (ALB) involves the assignment of various tasks of an assembly process to workstations, so as to optimise an objective function while the precedence constraints imposed on the set of tasks are satisfied and the execution time of each workstation does not exceed the cycle time. This problem belongs to the NP-hard class of combinatorial optimisation. Different variants of the ALB problem and solving techniques can be found in [1], [2].

In many practical situations, more complicated constraints caused by different technological factors may exist. The ALB problem considered in [3] includes a preplanned imbalance and assigns the tasks to particular types of workstations. A distinction of the set of tasks according to the process design in ”fixed” tasks and ”float” tasks is considered in [4].

Keywords and phrases: assembly line balancing, hybrid genetic algorithm, compatibility constraints, greedy method
(2000) Mathematics Subject Classification: 68T20, 90-08

This paper considers the deterministic ALB problem with compatibility constraints, assuming that the value of the cycle is predefined and the objective is to minimize both the number of the workstations and the smoothing index. An order-based genetic algorithm (GA) is presented for solving this problem. It is hybridised with a greedy method proper for the problem [5]. In section 2, the statement of the problem is presented. Section 3 outlines the greedy method for assembly line balancing with compatibility constraints. Section 4 explains the components of the GA, including the grafting of the greedy method. The results of experimental investigation are given in section 5. The paper ends with some final conclusions.

II. PROBLEM STATEMENT

The mathematical formulation of the deterministic ALB with compatibility constraints (shortly ALBC) can be stated as it follows ([6], [7]). Let be $V = \{1,2,...,n\}$ the set of the tasks of an assembly process. The acyclic digraph $G = (V, A)$ is associated to the precedence restrictions in task execution. If $(x, y) \in A$, then the execution of task y can begin after the task x is finished. The positive real number t_i is the execution time of task i, $i \in V$. Let us denote by C the pre-specified value of the cycle time and suppose that $t(i) \le C$, $i = 1,...,n$. Let be $Q = \{Q_1,...,Q_p\}$ a cover of i.e. $Q_i \subseteq V$, $i = 1,...,p$ and $V = Q_1 \cup ... \cup Q_p$. Let S be the set of all partitions $W = \{W_1,...,W_m\}$ of V, which satisfies the following conditions:

(1) $$\text{if } (x, y) \in A,\ x \in W_r,\ y \in W_s \Rightarrow r \le s\ ;$$

(2) $$T(W_j) = \sum_{x \in W_j} t(x) \le C,\ j = 1,...,m\ ;$$

(3) $$(\forall)\, j \in \{1,...,m\}\ (\exists) k(j) \in \{1,...,p\} \text{ so that } W_j \subseteq Q_{k(j)}.$$

Each member of S is called a *solution* to the ALBC problem. The solution with a minimum number of workstations m is an *optimal solution*. The condition (3) reflects the compatibility constraints. Since the classical ALB problem is NP-hard, the ALBC problem is at its turn NP-hard.

The concept of assembly line balancing problem with compatibility constraints is introduced in [5] as a tool for handling different technological restrictions and it is obtained from the originally formulated problem by adding the condition that the partition of the set of tasks into stations is compatible with a given cover of this set. In [5] it is shown how five modifications to the original problem can be treated in a rigorous and unitary manner as compatibility constraints: requirement of each station to contain a

limited number of types of equipments, requirement of tasks to be assigned to particular types of stations, the execution of some tasks in only a left (right)-of-line station, the association of tasks according with tasks skill level, and the separation of some tasks. A dynamic programming approach is proposed for solving ALBC problem, but even if it guarantees the solution optimality, the computing time needed for solving real life problem instances is prohibitive.

An effective tool for solving large ALBC problem instances is described in the next section. Moreover, the proposed GA can simultaneously treat many compatibility constraints addressed to different covers of the set of tasks.

III. GREEDY METHOD FOR LINE BALANCING WITH COMPATIBILITY CONSTRAINTS

The idea of the greedy algorithm is to create a solution $W = \{W_1,...,W_m\}$ by assigning tasks in a serial manner such as W_{j+1} is created after W_j, $j = 1,...,m-1$. The creation of workstations W_j is performed by indicating an optimizing measure that establishes the order of tasks that candidate to fill the current workstation. So, a list L of assignable tasks is created. This list contains unassigned tasks that have no predecessors or whose predecessors have already been assigned to the workstations under construction. List L is sorted in decreasing order of the processing times then by the number of sets of the cover containing the current task. The tasks in L are considered for the assignment in the resulted order. The algorithm starts by the initialization of workstation W_1 with the first task in list L that, at this moment, contains tasks without predecessors. If W_j is the current workstation then the first task y^* in L so that (i) it fits in the idle time of W_j and (ii) $W_j \cup \{y^*\} \subseteq Q_k$ for some $k \in \{1,....,p\}$ is added to W_j and list L is updated. If such a task does not exist, then a new workstation W_{j+1} is created. The process continues until the list L becomes empty and this happens if the digraph G is acyclic. If G is reduced to or contains a Hamiltonian path then this procedure becomes that used to compute the fitness of a chromosome in the genetic algorithm described in the next section. In this case, list L contains exactly one task at every time when the current workstation is under completion.

IV. HYBRID GENETIC ALGORITHM FOR ALB WITH COMPATIBILITY CONSTRAINTS

In this section the main components of the proposed hybrid GA are presented.

Solution representation. The individuals in the populations are represented as permutations of V, $x = (x_1,...,x_n)$ that sort V in topological order. Each chromosome represents a *feasible assembly flow*. This approach is a constructive one, the solution structure being calculated during the fitness evaluation and it is not stored during the evolution. In this way, the GA intrinsically operates with sets of solutions corresponding to the current chromosome instead of working with individual solutions. From this set of solutions that can be build on the corresponding assembly flow, the best one only gives the fitness value of the chromosome.

Initial population. The initial population is generated with the classical topological sorting algorithm that is adapted for randomly selecting the vertices forming the components of the returned topological sorting. The solution produced by greedy method described in the previous section is injected in the initial population. The population size as an empirically determined function of the number n of tasks is given in Table 1.

Table 1 - *Population size as a function of task number n*

n	8-30	31-60	61-100	101-200	201-300	301->
pop_size	40	53	66	106	130	200

Fitness function. Consider a chromosome $x = (x_1,...,x_n)$ and the solution $\{W_1,...,W_m\}$ constructed on this assembly flow. The procedure for computing the workstations corresponding to a given chromosome $x = (x_1,...,x_n)$ has the following steps.

Step 1. $m = 1$, $W_1 = \varnothing$

Step 2. for $i = 1,..,n$ do:

if ($T(W_m \cup \{x_i\}) \le C$) and ($W_m \cup \{x_i\} \subseteq Q_h$ for some $h \in \{1,...,p\}$)

then $W_m = W_m \cup \{x_i\}$

else $m = m + 1$, $W_m = \{x_i\}$.

This procedure ensures all data the fitness computation needs. Using this solution $\{W_1,...,W_m\}$ the fitness value for x is defined by the formula:

$$fit(x) = w_1 \cdot (m \cdot C - T) + w_2 \cdot \left[\frac{1}{m} \sum_{j=1}^{m} \left(C - T\left(W_j\right) \right)^2 \right]^{1/2},$$

where m is the number of workstations and $T_{tot} = \sum_{1}^{n} t_i$ is the total execution time. The first part of this function expresses the total idle time of the assembly line, whilst the second part is the smoothing index, that measures the

equality of the work distribution between workstations. The factor w_1 linearly varies from $w_{\max} = 0.9$ to $w_{\min} = 0.2$ when the number t of the evolution stages goes from 1 to $t_{\max}$, where $t_{\max}$ is the maximum number of stages, whilst $w_2 = 1 - w_1$.

Example. Let consider the digraph given in Fig. 1. The task times and the value of the cycle time are indicated in Table 2.

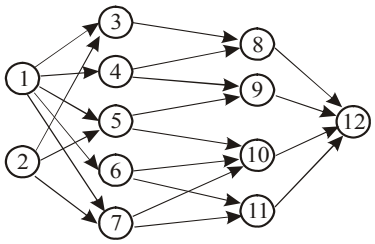


Fig. 1. The digraph of precedence restrictions

Table 2. The processing times of the tasks

Task	1	2	3	4	5	6	7	8	9	10	11	12	C
Time	4	5	2	2	3	4	2	1	2	2	1	0.5	10

Suppose that $Q_1 = \{1,2,3,4,5,8,9,6,7,10\}$ and are $Q_2 = \{9,6,7,10,11,12\}$ the cover sets and consider the topological sorting given by the chromosome $x = (1,2,3,4,5,8,9,6,7,10,11,12)$. By applying the previous procedure it results that $m = 3$, $W_1 = \{1,2\} \subset Q_1$, $T(W_1) = 9$, $W_2 = \{3,4,5,8,9\} \subset Q_1$, $T(W_2) = 10$, $W_3 = \{6,7,10,11,12\} \subset Q_2$, $T(W_3) = 9.5$ and $T_{tot} = 28.5$. Finally, it is obtained that

$$fit(x) = w_1(3 \cdot 10 - 28.5) + w_2\left((1^2 + 0.5^2)/3\right)^{1/2} = 1.5w_1 + 0.646w_2 .$$

Mutation. The mutation is applied to the entire population and each individual supports mutation with the same probability π_m. Once an individual is selected for mutation the basic mutation step is recursively applied ρ times. The value of multiplicity ρ depends on the number of chromosomes and the number of the current iteration and is defined by

$$\rho(t) = \begin{cases} ceil(a * t + b), & a * t + b > 0 \\ 1, & a * t + b \le 0 \end{cases}$$

where a = (1- *n/8*)/(0.9 $*$ $t_{\max}$ - 1), b = *n/8* - (1 - *n/8*)/(0.9 $*$ $t_{\max}$ - 1) and $ceil(x) = \min\{y \text{ integer} / x \le y\}$. Appropriate values for p_m are between 0.1 and 0.2. A mutation operator that preserves topological sorting is used.

Suppose that $v=(v_1,v_2,...,v_n)$ is a randomly selected chromosome. The basic mutation step tries to move a task v_h between the margins allowed by the precedence constraints. So, the vertex v_h, $h\in\{1,...,n\}$ is randomly selected and let us denote by v_l the rightmost predecessor of v_h and by v_r the leftmost successor of v_h. The mutation is successful either $l<h-1$ or $r>h+1$. If it is not, a next mutation attempt is made. If $r-h>h-l$, then the task v_h is moved to the position $r-1$, and the sequence $v_{h+1},...,v_{r-1}$ shifts leftwards one position. If $h-l\geq r-h$, then the task v_h is moved to the position $l+1$ and the sequence shifts rightwards. By moving the task v_h to the furthest position, the Hamming distance between the initial chromosome and its mutant is maximized, favoring a better variability of individuals. The performance of this operator is investigated in [9] and [10].

Crossover. For each crossover, two different parents are randomly selected from the matting pool that consists in the best s% individuals of the population, where $40\leq s\leq 60$. The crossover probability is π_c. Both parents are cut into a random number of parts r, where $2\leq r\leq n*C/T$. The first (second) offspring takes the first part from the first (second) parent and then the parts of both parents are alternated so that each task appears just once. This crossover operator ensures that the offsprings are topological sorted if so were their parents.

Hypermutation. As mentioned before, the first way of hybridisation is to put the solution produced by the greedy method into the initial population. In order to improve the performance of the GA, the greedy method is grafted on a mutation type operator. The result is a hypermutation operator that acts with the probability π_H (usually, $0{,}05\leq\pi_H\leq 0.08$). Two indexes j_1 and j_2 of workstations with $0<j_2-j_1\leq m/5$ are randomly chosen in the solution corresponding to the current chromosome. The greedy method is then applied to the ALBC instance corresponding to the tasks within this segment. The returned sequence of tasks replaces the former one in the chromosome.

Population management. The evolution is organized in stages. During the current stage, mutation, crossover and hypermutation produce new individuals that compete with current population for the next population. The survival selection is deterministic and elitist. The population passes from one evolution stage to the next stage until a stopping condition is reached. Denote by fit_k^* the best fitness value at the end of stage k-th evolution and let tol be a

prescribed tolerance threshold, If $(fit_k^* - fit_{k-1}^*)/fit_k^* \le tol$ along a prescribed number n_s of successive stages (usually, between 5 and 10), then this situation is assimilated with stagnation and the probability of the simple mutation is increasing with $s_m = 0.02$. If the fitness of the best individual continues to stagnate after this reinforcement of the mutation, for the next $2 * n_s$ stages, then it is considered that the best solution was reached and the algorithm stops. Another ending criterion is to achieve the maximum number of iterations, $t_{\max}$.

V. EXPERIMENTAL RESULTS

In order to prove the efficiency of the hybrid GA, an experimental investigation was carried out on classical test problems.

5.1 Generating of the cover sets

In the first phase of the experiment, some difficult problem instances have been produced using the test problems presented in [10] and [11] for ALB problem. The following procedure was applied.

Step 1. An instance of an ALB problem without compatibility constraints is considered. The hybrid GA developed in [6] is applied on this instance and from the best found assembly flow $x = (x_1,...,x_n)$ the corresponding solution $\{W_1,...,W_m\}$ is obtained. The cover sets $Q_1,...,Q_p$ are generated with the next steps:

1.1 For $i \in \{1,..,m\}$, let l_i and l_{i+1} be the limits of the workstation W_i a, i.e. $W_i = \{x_{l_i}, x_{l_i+1},..., x_{l_{i+1}}\}$;

1.2 Two random numbers $h_i \le l_i$ and $h_{i+1} \ge l_{i+1}$ are generated, $p = m$ and $Q_i = \{x_{h_i},..., x_{h_{i+1}}\}$, $i = 1,..., p$.

Step 2. This compaction step reduces the number of sets in the cover. A random position j is generated between 2 and p and the reunion of Q_{j-1} and Q_j is done and take $p = p - 1$. The reunion of the sets continues until the desired number of sets until $p \le 3/4 \cdot m$.

5.2 Performance report

The aim of the experimental investigation was to determine the distribution of the difference between the number of workstations computed by the algorithm and the corresponding lower bound defined as $m\_inf = ceil(T_{tot}/C)$.. The test data contained in LUTZ1_coverS.xls, ARC111_coverS.xls and SCHOLL_coverS.xls that can be found in [13].

For each value of the cycle time, repeated runs of the hybrid genetic algorithm are performed. For the solutions obtained in this way, the following situations are extracted:

(i) The characteristics of the best solution corresponding to the recalculation of the cycle time to the value of the maximal time of a workstation (*C_calc*) from the best solution given by the algorithm in 10 executions.

(ii) The characteristics of the solution with minimum number of workstations (*m_calc*), given by the algorithm in 10 runs.

(iii) The characteristics of the best solution given by the algorithm in 10 runs related to the product between *C_calc* and *m_calc* (*m·C* - the capacity supply of the line.

(iv) The characteristics of the best solution given by the algorithm in 10 executions related to the smoothing index (*smooth_index*).

(v) The characteristics of the best solution given by the algorithm in 10 executions related to the fitness (*fit* - the fitness being calculated with *C_calc* and *m_calc*).

(vi) The characteristics of the best solution given by the algorithm in 10 executions related to the balancing index (*balance_index*), given by $T/m \cdot C$.

Now, consider the LUTZ1_coverS.xls problem with 32 tasks ([13]). The graph of precedence constraints and the execution times of tasks are given in Fig. 2. The instances involve 6 different values for the cycle times. For $C = 2357$ and the lower bound *m_inf* = 6, the structure of the best solution produced by the hybrid GA is illustrated in Table 3.

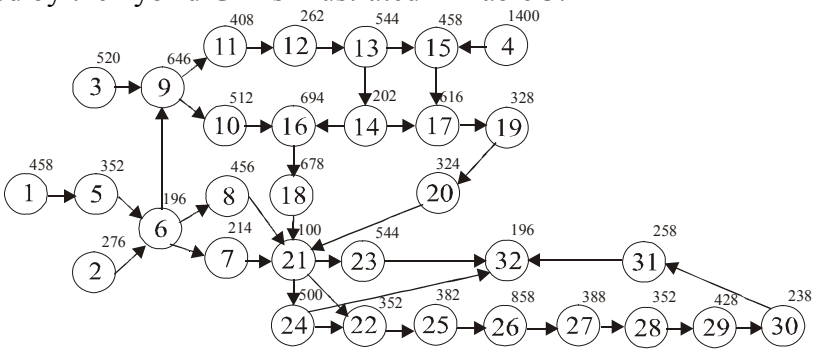


Fig. 2 - The digraph of the instance in LUTZ1_coverS.xls

In the last column of the Table 3, the cover sets including the current workstation are indicated.

Table 3 - *Structure of best solution for LUTZ1_coverS.xls with* $C = 2357$

Work-station W_i	Assembly flow in W_i	$T(W_i)$	Sets including W_i
W_1	4, 3	1920.00	Q_2
W_2	1, 5, 2, 6, 9	1928.00	Q_2

W_3	8, 7, 11, 12, 10	1852.00	Q_1
W_4	13, 15, 14, 17, 19	2148.00	Q_1
W_5	16, 18, 20, 21, 22	2148.00	Q_3
W_6	25, 26, 24, 27	2128.00	Q_3, Q_4
W_7	28, 23, 29, 30, 31, 32	2016.00	Q_5

The cover sets obtained with the procedure described in section 3.1 for $C = 2357$ are presented in Table 4.

Table 4 - *The cover sets for LUTZ1_coverS.xls,* $C = 2357$

Sets	Tasks
Q_1	7, 8, 9, 11, 12, 13, 14, 10, 15, 17, 19, 16, 18, 20
Q_2	4, 3, 1, 5, 2, 6, 7, 8, 9
Q_3	15, 17, 19, 16, 18, 20, 21, 22, 25, 26, 24, 27
Q_4	21, 22, 25, 26, 24, 27, 23, 28, 29
Q_5	24, 27, 23, 28, 29, 30, 31, 32

For $C = 2357$, the parameters used in the algorithm execution and the values of the target values obtained after 10 runs of the algorithm are presented in Table 5. This table includes: the lower bound calculated on the number of workstations (*m_inf*); the *best known* value (*m_best*) for no compatibility constraints; p_m – mutation probability; s – is the size of the population front the crossover is performed on; and the number of iterations up to the convergence detection (*it_conv*). The last column in Table 5 shows the values of the mutation probability after the detecting of the stagnation and the subsequent attempt to avoid this stagnation by increasing mutation probability. Column *it_ conv* gives the number of iteration when the first stagnation was detected, so algorithm actually stopped after more 20 generations. This table offers the distribution of *m_calc-m_best* in the last two rows.

Table 5 - *Data and target values of LUTZ1_coverS.xls instance for* $C = 2357$

Graph_code	No. of tasks							
LUTZ1_coverS.xls	**32**							
C/Target values	***m_inf***	***m_ best***	***p_m***	***fc***	***tol***	***inject y/n***	***mutH y/n***	
2357	6	6	0.1	0.65	0.0006	y	y	
Cases	***C_***	***m_***	***m·C***	***smooth***	***fit***	***balance***	***it_***	***p_m***

	calc	***calc***		***ind***		***ind***	***conv***	
(i) Sol. with *C_calc*	2148	7	15036	355.79	2047.36	0.94	120	0.20
(ii) Sol. with *m_calc*	2148	7	15036	355.79	2047.36	0.94	120	0.20
(iii) Sol. with *m·C*	2148	7	15036	355.79	2047.36	0.94	120	0.20
(iv) Sol. with *smooth_ind*	2148	7	15036	348.37	1944.8	0.94	228	0.30
(v) Sol. with *fit*	2148	7	15036	348.37	1944.8	0.94	228	0.30
(vi) Sol. with *balance_ind*	2148	7	15036	355.79	2047.36	0.94	120	0.20
Distribution *m_calc-m_best*	0	1	>=2					
Rel_freq.	0.2	0.7	0.1					

The number of workstations computed by the algorithm without compatibility constraints is denoted by m'. The relative frequencies of the values *m_calc - m_inf* and *m_calc-m'* are shown in Table 6 and correspond to the six values of the cycle time, namely 1583,1780,2035,2374,2848,3560, for each value 10 runs being executed. The values of the relative frequency range between 0 and 2. The most frequent value is 1. The pie graph of this estimation is shown in Fig. 3. The distribution *m*-m'* from second line in Table 6 indicates that in 85% of cases, the solution given by the GA without compatibility constraints is the same with the best solution. The comparison between these values indicates that the hybrid GA with compatibility constraints gives good results.

Table 6 - Distribution of m^*-m_0 and m^*-*m'* for LUTZ1_coverS.xls, *C* =1583, 1780, 2035, 2374, 2848, 3560

Relative frequency	0	1	>=2
m_calc - m_inf	0.84	0.1	0.06
m_calc - m'	0.85	0.15	0

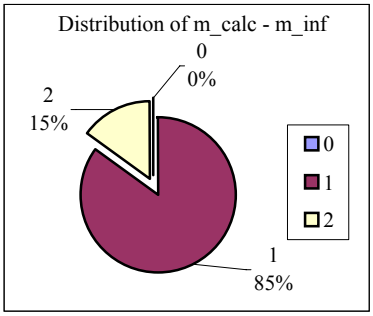


Fig. 3 - Distribution of m^*-m_0 for LUTZ1_coverS.xls

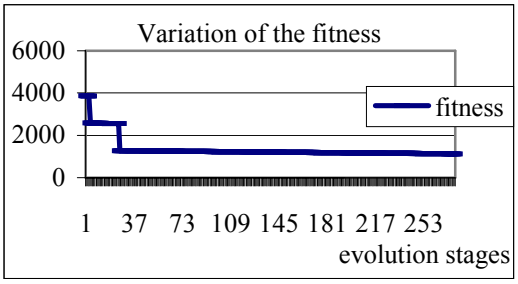
From the obvious inequality $m\_calc - m\_inf \leq m\_calc - m'$ it results that the real performance is better than the estimation above presented.

For all the six values of the cycle time, the minimal, average and maximal values of smoothing index, balancing index, and fitness are computed. These values are presented in Table 7.

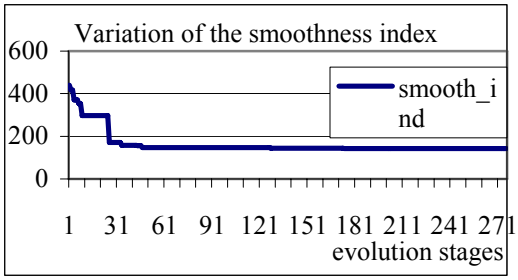
Table 7 - Values of the performance indexes and the cycle times for LUTZ1_coverS.xls]

C		1583	1780	2035	2374	2848	3560
m_0	min.	10	9	8	7	6	5
Smoothing index	min.	141.76	179.49	209.73	259.7	348.37	483.71
	av.	156.51	278.73	214.80	265.41	353.25	492.54
	max.	271.05	303.23	219.86	279.55	355.88	518.89
Balancing index	min.	0.84	0.87	0.92	0.90	0.93	0.86
	av.	0.91	0.88	0.94	0.94	0.937	0.92
	max.	0.92	0.92	0.94	0.95	0.94	0.94
Fitness	min.	1123.40	1309.3	1455.29	1580.42	1944.80	2367.03
	av.	1288.82	2392.42	1511.51	1709.58	2028.57	2435.37
	max.	2432.60	2729.96	1543.48	1760.70	2059.01	2473.08

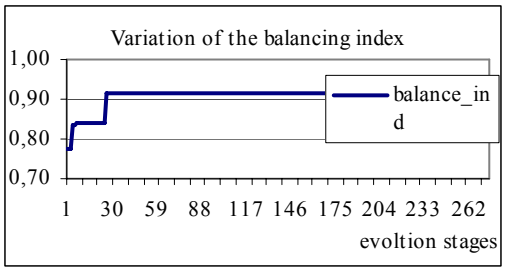
The typical variation of fitness, smoothing index and balancing index during the evolution of the considered example for $C = 2374$ is shown in Fig. 4 (a)-(c), respectively.



(a)



(b)



(c)

Fig. 4 - Variations of the fitness, smoothing and balancing indexes

The experimental tests using ALB instances with compatibility constraints obtained from ARC111_coverS.xls problem instance and the 297-task SCHOLL_coverS.xls problem ([13]) and the results obtained to corresponding simple ALB problems (without compatibility constraints) showed that the hybrid GA with compatibility constraints gives good results for medium and large size instances.

VI. CONCLUSIONS

In this paper, the ALB problem with compatibility constraints, a model requiring the compatibility of each solution to a given cover of the set of tasks, was treated. For solving this problem, an order based GA combined with a greedy technique had been designed. Computational experiments showed that the procedure is effective in finding good solutions. The performance comparison with the GA ignoring the compatibility constraints proved that the proposed approach is very promising.

The inclusion of the compatibility constraints into a successful evolutionary approach allows the solving of more complex practical problems. The proposed method is a general framework for treating practical applications in which different restrictions such as skill levels, task separation and left/right sided tasks are required.

References

[1] Scholl, A., **Balancing and sequencing of assembly lines**, 2nd ed., Physica-Verlag, Heidelberg, 1999

[2] Brudaru, O., Copaceanu, C., Valmar, B., **Assembly line balancing paradigm: Achievements and new trends**, in: New Trends in Computer Engineering. Technical University ''Gh. Asachi" Iasi, Polirom Press, Iasi, 187-218, 2003

[3] Johnson, R. V., **A branch and bound algorithm for assembly line balancing problems with formulation irregularities**, Management Science, 29, 1309-1324, (1983)
[4] Park, K., Park, S., Kim, W., **A heuristic for an assembly line balancing problem with incompatibilities, range, and partial precedence constraints**, Computers Ind. Eng, 32 (*2*), 321-332, (1997)
[5] Brudaru, O., **Assembly line balancing with compatibility constraints**, Econ. Comput. Econ. Cybern. Stud. Res. 27, No.1-4, 59-65 (1993)
[6] Brudaru, O., Copaceanu, C., Popovici, D., **A new hybrid genetic algorithm to "I"- assembly line balancing problem**, to appear in Bul. Inst. Polit. Iaşi, S. Matematica- Mecanica teoretica- Fizica, tom. LVI(LX), *1*, (2010)
[7] Brudaru, O, **Fuzzy compatibility constraints in assembly line balancing**, in Z i m m e r m a n n, H. J. (ed.): Proceedings of EUFIT'98–6th European Congress on Intelligent Techniques & Soft Computing. 3, Verlag Meinz, 1651-1655, (1998)
[8] Brudaru, O., **A genetic algorithm for assembly line balancing with compatibility constraints using a control mechanism based on information energy,** EUFIT'99 - 7th European Congress on Intelligent Techniques and Soft Computing, Aachen, Germany, Sept. 13-16, (1999)
[9] Brudaru, O. et al., **A mutation operator preserving the topological sorting. Part I**, Bul. Inst. Polit. Iaşi, XLIX(LIII), *3/4*, S. Textile. Pielărie, 98-100, (2003).
[10] Brudaru,O. et al., **A mutation operator preserving the topological sorting. Part II**, Bul. Inst. Polit. Iaşi, L(LIV), *1/2*, S. Textile. Pielărie, 115-124, (2004).
[11] http://www.assembly-line-balancing.de/files/uploads/SALBP data sets.zip
[12] http://www.misp.tuiasi.ro/obrudaru/line_balancing/SALB.rar
[13] http://www.misp.tuiasi.ro/obrudaru/line_balancing/CALBs.rar

OCTAV BRUDARU
Institute of Computer Science, Romanian Academy, Jassy Subsidiary
"Gh. Asachi" Technical University Jassy, Department of Management and Production Systems Engineering
e-mail: brudaru@tuiasi.ro
DIANA POPOVICI
Institute of Computer Science, Romanian Academy, Jassy Subsidiary
CINTIA COPACEANU
"Gh. Asachi" Technical University Jassy, Department of Management and Production Systems Engineering