Cellular Genetic Algorithm with Communicating Grids for Assembly Line Balancing Problems

Octav BRUDARU[1,2], Diana POPOVICI[1], Cintia COPĂCEANU [2]
[1]*Institute of Computer Science, Romanian Academy, Iaşi Subsidiary, Romania*
[2]*Gh. Asachi Technical University Iaşi, Romania*
brudaru@tuiasi.ro

***Abstract*—This paper presents a new approach with cellular multigrid genetic algorithms for the "I"-shaped and "U"-shaped assembly line balancing problems, including parallel workstations and compatibility constraints. First, a cellular hybrid genetic algorithm that uses a single grid is described. Appropriate operators for mutation, hypermutation, and crossover and two devoration techniques are proposed for creating and maintaining groups based on similarity. This monogrid algorithm is extended for handling many populations placed on different grids. In the multigrid version, the population of each grid is organized in clusters using the positional information of the chromosomes. A similarity preserving communication protocol between the clusters placed on different grids is introduced. The experimental evaluation shows that the multigrid cellular genetic algorithm with communicating grids is better than the hybrid genetic algorithm used for building it, whereas it dominates the monogrid version in all cases. Absolute performance is evaluated using classical benchmarks. The role of certain components of the cellular algorithm is explained and the effect of some parameters is evaluated.**

***Index Terms*—cellular genetic algorithms, communicating grids, I/U-shaped assembly lines, parallel workstations, and task compatibility constraints.**

I. Introduction

Genetic algorithms (GA) have demonstrated many advantages both for the algorithm designer and the beneficiary of the final results. They can be easily adapted to the required task and can be combined with other optimization techniques leading to hybrid methods. They easily imitate the human operating ways and incorporate them into specific operators. GA can be combined with machine learning concepts for obtaining a more intelligent behavior and they can be organized in different ways for a better exploiting of the intrinsic parallelism and for achieving better solution quality.

The recent literature [8] shows how broad and diverse the set of applications of these methods is and how promising the directions of treating complicate real problems are. In particular, evolutionary computation provides efficient and robust strategies to search for good solutions that fit to various constraints and objectives appearing in assembly line balancing problems ([4], [10]).

This paper presents a new type of cellular GA (CGA) that evolves many populations placed on different bidimensional grids. Specific procedures for mutation, crossover and survival are proposed for one toroidal grid. Placing rules for the offsprings and moving rules for the predators induce an intrinsic aggregation of the individuals with similar structure. The subpopulation within each grid is structured into clusters based on proximity. The clusters in different grids whose dominant individuals have similar features communicate between them. A similarity preserving communication protocol is proposed. This cellular multigrid genetic algorithm is used for solving assembly line balancing with prescribed cycle time. It addresses both the "I" and "U"-shaped lines serial and parallel workstations and compatibility constraints. The optimization goal is twofold: minimizing the number of workstations and smoothing their work times

Section II presents the simple assembly line balancing problems and its variants with compatibility constraints, parallel workstations and considers both I- and "U"- shaped lines. The components of the multigrid cellular genetic algorithm are described in Section III. Section IV presents the results of experimental investigation of the performance of the proposed algorithm. Last section summarizes the work.

II. Assembly Line Balancing Problem

Assembly line balancing (ALB) is a combinatorial optimization problem appearing in the design of synchronous assembly systems. Basically, it concerns the grouping of a set of tasks into workstations so that they have about the same processing time and the precedence constraints are satisfied. A comprehensive monograph of this paradigm can be found in [9].

Let be the acyclic graph $G=(V,A)$, where $V=\{1,2,...,n\}$ designates the set of tasks. The set of arcs $A\subset V\times V$ represents the precedence constraints. For each $i\in V$, $t_i>0$ is the execution time of task i. Let C be the prescribed cycle time. Let us denote by $W=\{W_1,...,W_m\}$ a partition of V into workstations. W is a solution to the *simple assembly line balancing* (SALB) problem if

$$T(W_j)=\sum_{i\in W_j} t_i \le C,\ j=1,...,m \tag{1}$$

and

$$\text{if } (x,y)\in A,\ x\in W_r,\ y\in W_s \text{ then } r\le s \tag{2}$$

Let S be the set of those partitions W of V satisfying (1)-(2). A member $W^*\in S$ is an *optimal solution* if $card(W^*)\le card(W)$, $(\forall)W\in S$.

This problem is NP-hard. Since it is difficult to find optimal solutions of large size instances by exact methods, various heuristic approaches have been developed. The GAs [10] are considered as a very effective search technique in solving SALB and several of its variants ([4], [9]).

Digital Object Identifier 10.4316/AECE.2010.02015

SALB concerns serial workstations, but when some execution times exceed the cycle time, the line contains *parallel* workstations made by connecting many identical workstations in parallel ensuring a period of C time units even if the execution times of some tasks may exceed C. This variant of balancing problem is called PALB.

Another variant of SALB deals with the so-called *compatibility* constraints (CALB) introduced in [3]. In this case, a cover $K=\{K_1,...,K_p\}$ of V is given and in addition to conditions (1-2), it is required that for $(\forall) j \in \{1,...,m\}$, $(\exists) h_j \in \{1,...,p\}$ so that $W_j \subseteq K_{h_j}$. This type of constraints allow five modifications to the original problem to be treated in a rigorous and unitary manner: requirement of each station to contain a limited number of types of equipment, 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.

The previously defined variants of ALB concern the "I"-shaped line. When the line is folded so that the front of the line and its end are juxtaposed, then an "U"-shaped line is obtained and in this case some workstations contains tasks that are executed for different batches of products, whilst other workstation contains tasks applied for the same batch [1]. This problem is called for short, USALB. The main difference between the "I"-shaped lines and the "U"-shaped line models is concerned with the identification of tasks to be assigned to a station. In the former case, each task is assigned to a station only after its predecessors have been assigned, whereas in the latter one, available tasks are those where either predecessor or successor tasks are assigned ([1]). This layout allows a better utilization of the human operators and equipment along the assembly structure. Both parallel workstations and compatibility constraints can be considered for "U" –lines, too. The corresponding variants are named UPALB and UCALB, respectively. The concept of U-line is illustrated for the assembly process whose processing times are given in Table I. For the sake of simplicity, the precedence constraints are omitted and a feasible assembly flow, namely (1,2,6,7,11,4,3,5,8,9,10,12), is given in Fig. 1(a) for the I-shaped line.

TABLE I. THE PROCESSING TIMES OF THE TASKS

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

In Fig. 1(b), the same assembly flow is folded to form a U-shaped line.

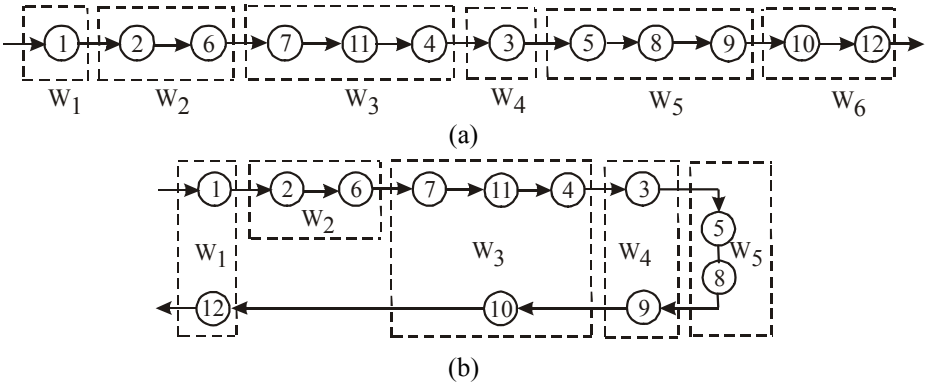

Figure 1. I-shaped line (a) and U-shaped line (b).

The proposed CGA described in the next section is able to solve all these variants of ALB problem.

I. MULTIGRID CELLULAR GENETIC ALGORITHM

In this section, the basic components of multigrid CGA (MCGA) for ALB are described. For the sake of clarity, the monogrid CGA is described first. Then the effect of introducing multiple communicating grids is analyzed.

A. Basic principles of CGA

A good tradeoff between exploration and exploitation is one of the key issues in the practice of GAs and the use of subpopulations based on the similarity is often a good decision ([2]). Multiple subpopulations and an adequate communication patern can lead to a good compromise between these search goals. This is the reason for which, depending on the problem, the cellular model could be better than other types of GAs. In a CGA the individuals are placed on a bidimensional lattice. The management resulting from this type of anchoring does not favor the multiplication of the best solutions and this avoids the premature convergence ([5], [6]).

The CGA proposed in this paper for a class of assembly line balancing problems combines the advantages of the grid mapping with those of segregative approach. It introduces new elements regarding the mapping of the individuals produced by genetic operators, it describes a new tactic for survival selection and it clusters the individuals using their positional information in view of communication. A feature function induced by the time profile of a solution is introduced together a similarity preserving communication protocol that improve the exploration-exploitation tradeoff.

B. Monogrid CGA

This monogrid version of CGA is an order-based GA together a proper management of the grid that support preys and predators. It can solve all variants of ALB presented in the previous section.

1) Chromosomes, their mapping on grid and evaluation

A solution (chromosome) to the problem is represented as

a topological sorting of V, $x = (x_1,...,x_n)$, that practically is the *assembly flow*. Consider the bidimensional grid as a square matrix $\Gamma = (\gamma_{ij})$ of $N \times N$ locations, where N^2 is larger than the size of the population P of the cellular algorithm. Usually, $N = 3 * card(P)$. If $x \in P$, then $pos(x)$ is the pair of coordinates (i, j) where x is placed on Γ and γ_{ij} represents the chromosome x or its address. The case $N = 5$ is illustrated in Fig. 2.

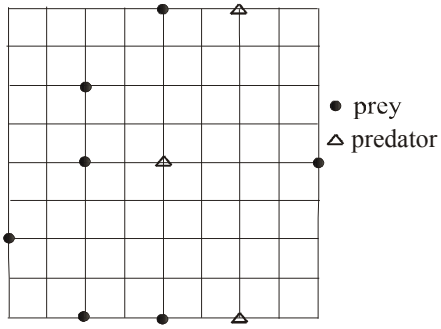

Figure 2. Two-dimensional grid with 7 preys (●) and 3 predators (Δ).

For avoiding the checking of limits, the components in $pos(x)$ are *modulo* N computed. Further, $\Omega((i, j), h)$ is the set of points in Γ placed on a square centered in (i, j) whose side length is $2h$. The initial population is randomly generated by a slightly modified version of the classical topological sorting algorithm and for instance it is randomly mapped on the grid.

The fitness value $f(x)$ associated to a chromosome x is defined by

$$f(x) = w_1 \cdot (mC - T_{tot}) + w_2 \cdot \left(\frac{1}{m} \sum_{j=1}^{m} \left(C - T(W_j)\right)^2 \right)^{1/2},$$

where $T_{tot} = \sum_{j=1}^{m} T(W_j)$. The first term in $f(x)$ represents the total idle time of workstations, whilst the second indicates the smoothness of the workstations. The weights w_1 and w_2, verify $w_1 + w_2 = 1$ and w_1 varies from 0.9 to 0.2 when the iteration number goes from 1 to the maximum number of evolution stages. Thus, at the beginning, the evolution is focused on the minimizing of the number m of workstation, whereas in its second part the goal is to reduce the imbalance between workstations.

The $f(x)$-value is computed by a simple algorithm that assigns tasks to workstations exactly in the order of tasks given by x. For "I"-lines, the tasks are considered from left to right, whereas for U-lines the assigning goes from the both ends of x towards an interior task in x ([1]).

The grid Γ is populated with predators that eliminate poor performance preys. The number of predators is denoted by R, where $R \approx 0.75 * card(P)$.

The evolution is organized in stages. During each stage, the current population supports mutation, hypermutation, crossover and devoration.

C. Genetic operators

Besides mutation and crossover, a hypermutation operator is used. This third operator grafts a greedy method on the GA.

1) Mutation operator

If $x = (x_1,...,x_n)$ is a chromosome, then let $\mu(x)$ be the mutation result. One selects a random task x_j in x and x_j is moved in a random position between the rightmost predecessor and the leftmost successor of x_j, so that the distance from the original position j is smaller as the evolution advances. For increasing its effect, this operator is applied recursively and $\mu^r(x)$ means the applying of μ, r times on it. The r–values linearly decrease from $n/10$ to 1 when the number of evolution stages goes from 1 to the maximum number of stages. The distance between $pos(x)$ and $pos(\mu^r(x))$ is proportional to r. In this way, if r is small then x suffers small changes, $\mu^r(x)$ is similar enough to x and $\mu^r(x)$ lies close to x. One tries to put $y = \mu^r(x)$ on a free position in $\Omega(pos(x), h)$, where h linearly decresses from 1 to $N/2$, for r varying from 1 to $n/10$. If such a free position does not exist, then a new attempt is made by increasing h with 1, and so on. Mutation enforces a similarity-based aggregation of chromosomes.

2) Hypermutation operator

The hypermutation incorporates a greedy method based on the *first fit decreasing time task* rule. For "I" -lines, this rule selects the candidates among the tasks with no/already assigned predecessors. If "U"-lines are considered, this rule suffers a minor modification in order to construct the technological flow simultaneously from both ends toward an interior position of the chromosome, If r and s are the cutting points in the chromosome x, then a new chromosome $h(x)$ is obtained by replacing the genes (tasks) between r and s of x with the assembly flow that is the solution produced by the greedy method applied to the problem instance only containing the tasks between r and s and for which the precedence constraints are given by the corresponding subgraph of G. The positions of $h(x)$ on the grid is given by the difference $s - r$. Namely, if $(i, j) = pos(x)$, then starting from (i, j), one searches a place for $h(x)$ in $\Omega((i, j), k)$, where k is a positive integer satisfying the following conditions:

(i) there is a free position in $\Omega((i, j), k)$,

(ii) $q(s - r) \le 4k(k + 1)$,

(iii) k is minimum,

where q is a parameter tuning the dispersion of the offsprings ($N/4 < q < N$) around the original prey. This is illustrated in Fig. 3. Larger population size requires larger values of q.

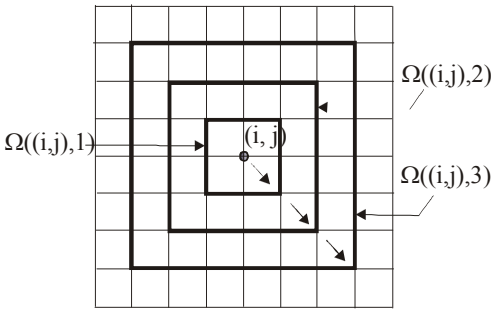

Figure 3. Successive larger boundaries to place the hypermutation result.

3) Crossover

The crossover uses the best 40% of the individuals as mating pool. The operator randomly generates ns cutting points on both parents in the same positions and alternately transfers the genes from the parents to each child so that the topological sorting of the offspring is preserved. The number ns decreases from $ns_{\max}$ to $ns_{\min}$ as the number of evolution stages varies from 1 to its maximum value, where $1 < ns_{\min} < ns_{\max} < n/(C/\max_i t_i)$. This variation of ns is adopted for protecting the constructive blocks whose length becomes higher during the evolution. If x and y are the parents and c_x and c_y are the resulted children, then one tries to put c_x and c_y to those points of Γ that are closest to the convex combinations $1/4\,pos(x)+3/4\,pos(y)$ and $1/4\,pos(y)+3/4\,pos(x)$, respectively. This is illustrated in Fig. 4. If these points are not free, than a new attempt is made, enlarging the vicinities.

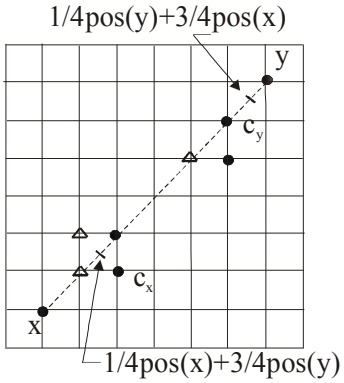

Figure 4. Placing the results of crossover on the grid.

Mutation, hypermutation and crossover are applied with probabilities π_m, π_H and π_c, respectively.

D. Survival selection

Survival selection (devoration) is made by the predators, which are randomly placed on the grid and move during the evolution in order to capture the poorest prey reached on their route. Each evolution stage ends with the devoration. The effect of different rules for driving their movement has been evaluated. The two best found rules are presented. Let (i, j) be the position of the current predator.

Rule d1. Let (x_g, y_g) be the gravity center of the set of all preys. If the predator does not find a prey in $\Omega((i, j),1)$ it moves to the nearest position in the direction from (i, j) to (x_g, y_g), as it is shown in Fig. 5.

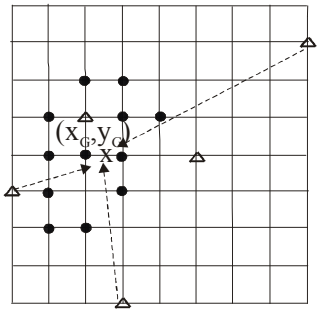

Figure 5. Predators moving to the center of gravity of the population.

Rule d2. The set of all preys is organized in c groups, applying, for example, the c-means algorithm to the set of positions $\{pos(x)/x \in P\}$. Let $\bar{x}_1,...,\bar{x}_c$ be the centroides of the resulted clusters that are not necessarily on Γ. If the current predator does not find a prey in its smallest vicinity it moves to the closest centroid until it finds one. This is shown in Fig. 6. In this way, the efficiency of hunting increases and the overload of the grid capacity is avoided, whilst the predators are distributed to the preys in a balanced way. If $card(P) \le 80$ then rule (d1) works better than (d2) otherwise (d2) dominates (d1).

The devoration acts until the number of preys reaches the prescribed population size. The survival selection made by devoration does not guarantee that the best solution survives. Whenever a best solution is devoured, its copy enters a thesaurus.

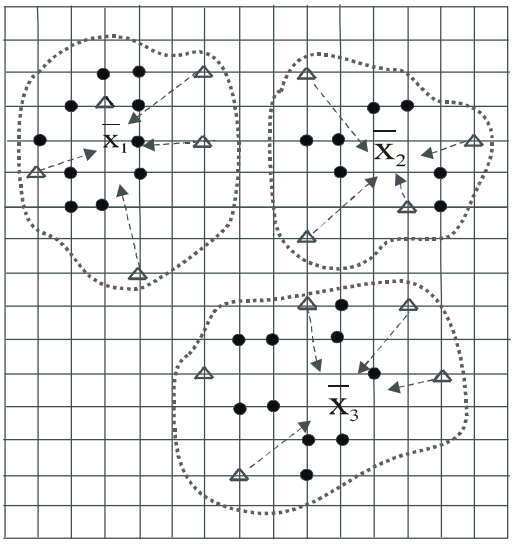

Figure 6. Predators moving to the nearest group of preys.

E. MANY COMMUNICATING GRIDS

The multigrid cellular algorithm uses M copies of the monogrid algorithm that act on the grids $\Gamma_1,...,\Gamma_M$ and communicate between them. The communication is activated at every r evolution stages. The population of preys in each grid Γ_i is organized in a number of K_i clusters, using the position vectors of the chromosomes, like in rule (d2). Whenever communication starts, K_i is selected so that the sum of squares of distances between the position vectors of the preys and the centroids is minimized and the size of a cluster is between 20 and 30. Actually, devoration rule (d2) uses the clustering solution computed for communication. Due to the action of mutation,

hypermutation and crossover operators, an intrinsic geographic clustering is produced and the solutions within each agglomeration of preys have a certain degree of similarity. This agglomeration is detected and exploited by the unsupervised clustering made for communication. Denote by C_{ij} the cluster j in the grid Γ_i, $j=1,...,K_i$, $i=1,...,M$. Take a chromosome x among the best 10% solutions in C_{ij} and let $W_1,...,W_m$ be the workstations built on x. Construct the so-called *feature vector* $\varphi(x)$ having a fixed number l of components,

$$\varphi(x)=(\frac{1}{m}\sum_{h=1}^{m}\left(1-\frac{T(W_h)}{C}\right),\frac{T(W_1)}{C},...,\frac{T(W_m)}{C},0,...,0),$$

where the first $m+1$ components can be computed during the calculation of fitness of x, whilst the remaining $l-(m+1)$ are set to zero, $m+1<l\le n+1$.

For each cluster C_{ij}, let Φ_{ij} be the average of the feature vectors computed for the best 10% individuals in C_{ij}. Consider the similarity threshold $\delta>0$. The value of δ is experimentally established in order to obtain a significant communication between grids. Clusters C_{ij} and C_{rs} ($i\ne r$) communicate if $\|\Phi_{ij}-\Phi_{rs}\|\le\delta$ and the copies of the best 10% of individuals move from C_{ij} to C_{rs} and vice versa. For the considered test instances, some appropriate values of the threshold δ are in the range $0.15\div0.35$. A prey coming into C_{ij} is positioned using the mutation placement rule in which the roles of x and $\mu^r(x)$ are played by a randomly selected prey in C_{ij} and the incoming prey, respectively. This communication scheme preserves the similarity of solutions within each cluster.

The activity of a grid stops when its best fitness stagnates for a prescribed number of successive stages. The activity of the multigrid cellular genetic algorithm stops when all grids stopped.

As compared to some existing cellular GAs, three improvements are proposed in this paper: (i) the grafting of a very efficient greedy method on the GA (as specific contribution to ALB algorithmics), (ii) management of placement/moving of the preys / predators on the grids so that solutions having similar structure organize themselves in compact regions and (iii) the introducing of multiple grids that interchanges solutions using a communication protocol that preserves the similarity in the feature space.

II. Performance Of The Multigrid Version

Further, the results of some experimental investigation of the performance of the MCGA are presented. The experiments addressed three assembly instances Lutz (32 tasks), Arcus (111 tasks) and Scholl (297 tasks). Starting from these instances initially proposed for SALB ([11], [14]), some difficult instances were produced for PALB and CALB, which can be found in [13] and [15], respectively. The C-values, both for "I" and "U"-shaped lines are shown in Table II. The four grids variant of MCGA was used.

TABLE II. C-values for test ALB instances

I/U	SALB, CALB	PALB
Lutz	1414, 1572, 1768, 2020, 2357, 2828	565,588,598,632,646, 678
Arcus	5757, 6016, 6540, 7162, 7916, 8847, 10027,11378, 17067	2878,2989,3067,3189, 3268,3375, 3390,3356,3498,3945
Scholl	1394, 1548, 2049, 2680	1394, 1422, 1452, 1483

A. Absolute error estimation

Further, $m*$ denotes the number of workstations returned by the MCGA, whilst m_0 is a computed lower bound of the minimum number of workstations associated to the respective instance. For each problem instance, a number of 30 runs were achieved. For I-shaped lines, the obtained distribution of the difference $m*-m_0$ is given in Table III.

TABLE III. Error distribution for "I"-shaped lines

	SALB			
Lutz	$m*$-m_0	0	1	
	rel. freq.	0.98	0.02	
Arcus	$m*$-m_0	0	≥1	
	rel. freq.	0.86	0.14	
Scholl	$m*$-m_0	0	1	2
	rel. freq.	0.15	0.50	0.35
	PALB			
Lutz	m*-m0	0	1	2
	rel. freq.	0.31	0.49	0.2
Arcus	m*-m0	0	1	2
	rel. freq.	0.16	0.41	0.43
Scholl	m*-m0	0	1	≥2
	rel. freq.	0.08	0.78	0.14
	CALB			
Lutz	m*-m0	0	1	≥2
	rel. freq.	0	0.95	0.05
Arcus	m*-m0	0	1	2
	rel. freq.	0.16	0.55	0.25
Scholl	m*-m0	0	1	2
	rel. freq.	0.21	0.74	0.05

Table IV shows the performance for "U"-shaped lines for the test data in [12] and [16].

TABLE IV. Error distribution for "U"-shaped lines

	USALB			
Lutz	$m*$-m_0	0	1	
	rel. freq.	0	1	
Arcus	$m*$-m_0	0	1	
	rel. freq.	0.13	0.87	
Scholl	$m*$-m_0	0	1	
	rel. freq.	0	1	
	UPALB			
Lutz	m*-m0	0	1	2
	rel. freq.	0.17	0.43	0.40
Arcus	m*-m0	0	1	2
	rel. freq.	0.25	0.59	0.16
Scholl	m*-m0	0	1	
	rel. freq.	0	1	

	UCALB			
Lutz	m*-m0	0	1	
	rel. freq.	0	1	
Arcus	m*-m0	0	1	2
	rel. freq.	0.27	0.45	0.28
Scholl	m*-m0	0	1	2
	rel. freq.	0.21	0.54	0.25

Table V offers a synopsis of the performance of MCGA for USALB and Scoll-297 for 4 different C-values. Notice the high values of the balancing index $I=T_{tot}/(m*C)$ in the last row.

TABLE V. MAIN PERFORMANCE MEASURES (USALB, SCHOLL-297)

C		1394	1548	2049	2680
m_0		50	45	34	26
smoothing term	min.	30.19	36.94	65.53	122.11
	av.	32.13	37.44	68.84	159.54
balancing index	av.	0.98	0.98	0.98	0.96
	max.	0.98	0.99	0.98	0.97

B. Comparison between MCGA and its non-grid version

Denote by BGA the basic (non-grid) GA obtained from the monogrid CGA by removing the management induced by the mapping of chromosomes on grid and using the deterministic elitist survival selection instead of devoration. The averages of the absolute error's distributions obtained in the experimental evaluation of MCGA and BGA are denoted by μ_{MCGA} and μ_{BGA}, respectively. These values are given in Table VI for ""I" and "U" –lines, the three types of problems and test data mentioned above. The quality index Q_{ind} is the ratio between μ_{BGA} and μ_{MCGA} and it measures how much better MCGA than BGA is. The greatest quality gain is 8 and it is obtained for SALB and Lutz (32 tasks) whilst for USALB and Arcus (111) both methods recorded the same performance.

TABLE VI. COMPARISON BETWEEN MCGA AND BGA

"I"-shaped:		**SALB**	**PALB**	**CALB**
Lutz	μ_{MCGA}	0.02	1.89	1.05
	μ_{BGA}	0.16	4.796	1.15
	Q_{ind}	8	2.5376	1.0952
Arcus	μ_{MCGA}	0.14	2.27	2.01
	μ_{BGA}	0.21	2.85	2.46
	Q_{ind}	1.5	1.2555	1.2239
Scholl	μ_{MCGA}	1.5	1.273	1.84
	μ_{BGA}	1.64	1.3	1.8
	Q_{ind}	1.0933	1.0212	0.9783
	av. of Q_{ind}	3.5311	1.6048	1.0991
"U"-shaped:		**USALB**	**UPALB**	**UCALB**
Lutz	μ_{MCGA}	1	2.23	1
	μ_{BGA}	1.21	3.34	1.02
	Q_{ind}	1.21	1.498	1.02
Arcus	μ_{MCGA}	0.87	0.91	2.01
	μ_{BGA}	0.87	1.16	5.04
	Q_{ind}	1	1.2747	2.5075
Scholl	μ_{MCGA}	1	1	1.96
	μ_{BGA}	1.17	1.13	3.84
	Q_{ind}	1.17	1.13	1.9592
	av. of Q_{ind}	1.1267	1.3008	1.8289

The averaged values of Q_{ind} over the three sets of test data are given for each problem in the last row of each "I" or U" -shaped case. The maximum quality gain 3.5311 is offered for SALB. Clearly, MCGA still dominates BGA. The gain is especially due to the many communicating processes that succesfully prevents premature stagnation. For Arcus (111 tasks), on Intel Atom 1.6 GHz, 1GB RAM, 280 evolution stages of BGA take about 25 s, whereas four grids require 400 s, for about 320 stages, the communication cost being included in this last time.

C. Other quality indicators

The variation of fitness related indicators obtained for two grids Γ_1, Γ_2 is shown in Fig. 7-9, for Arcus (111 tasks) instance with $C = 6837$, for which $m_0 = 22$ and $m^* = 23$.

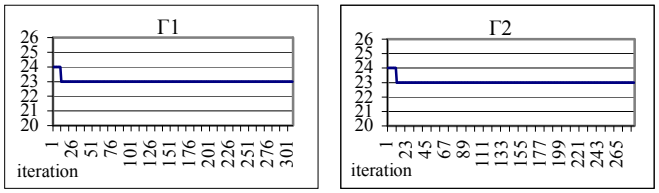

Figure 7. Number of workstations of the best-found solutions in Γ_1 and Γ_2.

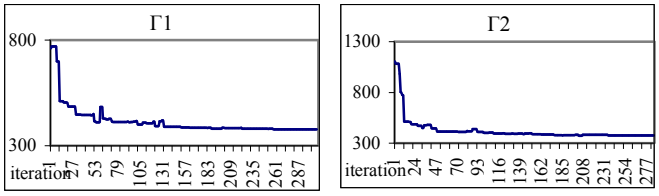

Figure 8. Smoothing index of the best-found solutions in Γ_1 and Γ_2.

At the beginning, the target is the minimization of the number of workstations (Fig. 7). Evolution continues with the minimizing of the smoothing term in fitness function (Fig.8).

The rapid variations of the balancing index (Fig. 9) when the number of workstations is constant is due to the setting of the current C value to the greatest load time of the workstations in the best found solution.

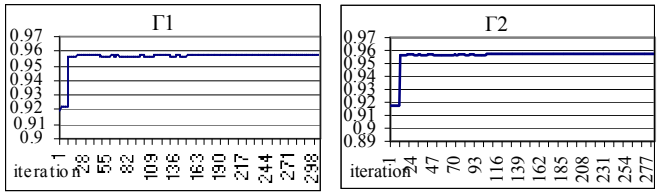

Figure 9. Balancing index of the best-found solutions in Γ_1 and Γ_2.

D. Communication between grids

The main features of the multigrid cellular approach are the multiple populations hosted by the grids and the type of communication between similar clusters in different grids.

The maximum of the total traffic appears after the first third of the evolution stages as a result of the creating and developing the clusters based on similarity. At the end of exploration the communication flow is modest due to the exploitation of current clusters when the variability within the clusters in each grid is diminished.

A large number of communicating clusters can be maintained by increasing the similarity threshold δ, but this delays the convergence detection with no improvement of the solution quality. The communication increases the local variability and this leads to a pseudo-periodic variation of average fitness in each grid.

The influence of the period of communication r on the solution quality and on computational effort was experimentally observed. The obtaining of a general rule for *a priori* determining r as a function of the parameters appearing in MCGA seems to be difficult.

A film showing the evolution and communication process of MCGA with 4-grids for solving the Arcus (111 tasks) instance [15] can be found at [17].

E. Some improvements

A better placement of the initial population can be done by computing the feature vector for each individual. A clustering of these images using a two-dimensional Kohonen network is applied and a simple heuristic maps the preys within each cluster to individual cells in compact regions of the grid. This improves the fitness of the best-found solutions with 6-7%.

The quality of the found solutions is better if the number of grids increases. If the number of grids is too large, the gain in quality becomes modest as compared to the computational effort. An empiric recipe is to increase the number of grids by 2 whenever the number of tasks augments with 100.

III. CONCLUSION

A new cellular genetic algorithm with communicating grids was proposed for efficiently solving a class of assembly line balancing problems. The algorithm contains efficient components and an appropriate management strategy that achieves a pseudo-segregative evolution of each grid. Experimental evaluation testifies its good performance. It offers both a fine- and large-grained parallelism and it can be easily extended to other optimization problems.

REFERENCES

[1] Ajenblit, D. A., Wainwright, R. L.: Applying genetic algorithms to the U-shaped assembly line balancing problem. In: Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, pp. 96-101, 1998.

[2] Alba, E., Troya, J. M.: An analysis synchronous and asynchronous parallel distributed genetic algorithms with structured and panmictic islands. In: IPPS/SPDP Workshops, pp. 248-256, 1999.

[3] Brudaru, O., Assembly line balancing with compatibility constraints, Econ. Comput. Econ. Cybern. Stud. Res. 27, No.1-4, 59-65 (1993).

[4] Gen, M., Cheng, R., Lin, L., Assembly Line Balancing Models, Network Models and Optimization, Springer London, 477-550 , 2008.

[5] Haynes, Th., Sen, S.: Evolving behavioral strategies in predators and pray. In IJCAI-95 Workshop on Adaptation and Learning in Multiagent Systems, pp. 32-37, 1995.

[6] Li, X., Suterhand, S., A.: Cellular Genetic Algorithm Simulating Predator – Prey Interactions. Technical Report, School of Comp. Sci. and Information Technology, RMIT University Melbourne, 2006.

[7] Michalewicz, Z., Genetic Algorithms + Data structures = Evolution Programs, Springer Verlag, Berlin 1994.

[8] Paszkowicz W. Genetic Algorithms, a Nature-Inspired Tool: Survey of Applications in Materials Science and Related Fields, Materials and Manufacturing Processes, vol. 24, 2, Feb. 2009, p. 174 - 197

[9] Scholl, A., Balancing and sequencing of assembly lines. 2nd edition, Physica-Verlag, Heidelberg.

[10] Tasan S., Tunali, S., A review of the current applications of genetic algorithms in assembly line balancing, Journal of Intelligent Manufacturing, Springer Netherlands, vol. 19, Number 1 / February, 49-69, 2008.

[11] Octav BRUDARU , http://www.assembly-line-balancing.de/files/uploads/ SALBP data sets.zip, accessed June 2007.

[12] Octav BRUDARU, http://www.assembly-line-balancing.de/files/uploads/UALBP-1 data sets.zip, accessed June 2007.

[13] Octav BRUDARU, http://www.misp.tuiasi.ro/obrudaru/ line_balancing/PALB.rar, posted Oct. 2009.

[14] Octav BRUDARU, http://www.misp.tuiasi.ro/obrudaru/ line_balancing/ SALB.rar, posted Oct. 2009.

[15] Octav BRUDARU, http://www.misp.tuiasi.ro/obrudaru/ line_balancing/CALBs.rar, posted Oct. 2009.

[16] Octav BRUDARU, http://www.misp.tuiasi.ro/obrudaru/ line_balancing/UALB.rar, posted Oct. 2009.

[17] Octav BRUDARU, http://www.misp.tuiasi.ro/obrudaru/ line_balancing/Cell-AG- Arcus 111-4-grids.avi, posted Oct. 2009.