BULETINUL Universităţii Petrol – Gaze din Ploieşti	Vol. LXI No. 1/2009	87 - 96	Seria Matematică - Informatică - Fizică

Mechanisms to Avoid the Premature Convergence of Genetic Algorithms

Elena Simona Nicoară

Universitatea Petrol-Gaze din Ploieşti, Bd. Bucureşti 39, Ploieşti, Catedra de Informatică
e-mail: snicoara @upg-ploiesti.ro

Abstract

The optimization by genetic algorithms often comes along with premature convergence bias, especially in the multimodal problems. In the paper, we propose and test two mechanisms to avoid the premature convergence of genetic algorithms by preserving the population diversity in two different manners. These are the dynamic application of many genetic operators, based on the average progress, and the population partial reinitialization. The mechanisms were tested by implementing them in the NSGA_II algorithm, applied to one of the most difficult job shop scheduling test problems, ft10. The comparative analysis between the new algorithm and the NSGA_II in the absence of the submitted mechanisms, alongside with an elitist and the canonic genetic algorithm, proves the usability of both proposed mechanisms.

Key words: *genetic algorithms, optimization, progress of the genetic operators, job shop scheduling*

Introduction

Over the last thirty years, the genetic algorithms and their hybrids have been applied to various optimization problems, by reason of their multiple advantages: free derivative characteristics, simple preparation of the optimization model, the parallel nature of the search etc. One crucial issue for the genetic algorithm success, especially for the difficult problems, is to avoid the premature convergence of the algorithm to suboptimal regions.

The premature convergence of a genetic algorithm arises when the genes of some high rated individuals quickly attain to dominate the population, constraining it to converge to a local optimum. In this case, the genetic operators can not produce any more descendents better that the parents (Fogel, 1994); the algorithm ability to continue the search for better solutions is therefore substantially reduced.

To avoid the premature convergence, in a genetic algorithm is imperative to preserve the population diversity during the evolution. In other words, the population diversity ensures avoiding the premature convergence. Among the methods used for this, we can enumerate: restricted selection, dynamic application of mutation, constraints for crossover and mutation probabilities, stochastic universal sampling, variable fitness assignment, population partial reinitialization, individuals grouping methods, restricted mating, elitism, symbiogenesis, species conserving techniques, ranking sort based on Pareto dominance, local search based on diversity. All these methods are heuristics by definition and their effects vary for different problems.

In the real world, there are difficult instances where no single method is adequate.

Another key aspect is the double character of the population diversity:

- o diversity in the objective space and
- o diversity in the parameters space.

For some problems is sufficient to preserve the diversity in one space; for others, a good genetic algorithm must maintain population diversity in both spaces.

Two imperatives are related to the premature convergence:

- o identify the occurrence of the premature convergence through various measures and
- o evaluate its extent.

The measures to detect the premature convergence are in fact measures for level of population degeneration. Srinivas and Patnaik (1994) use as measure the difference between the average fitness and the best fitness in population. Depending on this difference, they adaptively vary the crossover and mutation probabilities. The authors of [3] propose as statistical measures the average Hamming distance between individuals and the variance of Hamming distances, both independent on genes number and population dimension.

In the paper, we propose two mechanisms for maintaining the population diversity, both of them based on the average progresses of genetic operators during evolution.

Mechanisms to Avoid the Premature Convergence of the Genetic Algorithm

The submitted mechanisms are:

- o the dynamic application of crossover and mutation operators and
- o the population partial reinitialization.

Their goal is followed up in two different manners. The first one acts slowly, from beginning of evolution to the end of it. More precisely, two sets of operators (for crossover stage and for mutation stage) are used instead of two operators (one crossover operator and one mutation operator). At every generation, one operator in each set is dynamically applied, based on the selection probability, dependent on the average progress.

The population partial reinitialization is applied only when the risk of premature convergence appears. Formerly, the reinitialization was applied for parts of population, after a certain time or whenever the search stagnates. For the first time, Fonseca and Fleming insert in the population, at every generation, a small number of „imigrants", randomly generated.

The Dynamic Application of Crossover and Mutation Operators

The idea was developed because every genetic operator has an inherent behavior and, consequently, an inherent progress rate. Additionally, the different characteristics of problem instances determine different appropriateness levels for operators.

For a given genetic encoding, some operators work better at beginning of evolution and others after finding some good regions. In other words, some operators tend to better explore or better exploit the problem space (parameters space or objective space). Depending on the instance dimension and complexity, and based on experience on that instance, we may want to impose a certain exploration/exploitation ratio.

For the crossover operators, the author designed a formula for progress assignment, according to the specified exploration/exploitation ratio. This formula works for both uniobjective and multiobjective problems; it considers all the dominance relations between parents and descendents.

To compute the progress of a crossover operator *x* applied to the pair of parents (P_1, P_2), from which the descendent *D* is produced, we use the following formula:

$$\Pi(x)=\begin{cases}1, & \text{if } D \text{ dominates } P_1 \text{ and } P_2\\ \max(1-\frac{k_1 * t}{G},0.5), & \text{if } D \text{ dominates one parent}\\ & \text{and no dominance relation with the other parent}\\ \max(0.5-\frac{k_2 * t}{G},0), & \text{if } D \text{ is dominated by at least one parent}\\ 0.5, & \text{if no dominance relation exists between } D \text{ and } P_1, P_2\\ 0, & \text{otherwise}\end{cases}, \quad (1)$$

where *t* is the current generation, *G* is the maximum number of generations and k_1, k_2 the parameters which enforce the velocity of reducing the progress during the evolution (from 1 to 0.5 if *D* dominates one parent and, respectively, from 0.5 to 0 if *D* is dominated by at least one parent). Consequently, small values for k_1 and k_2 allow a wider exploration of the search space at the beginning of the evolution. The reason is that the algorithm forbids a massive proliferation for qualitative individuals.

The third condition in formula (1) is met when *D* is dominated by both parents, or *D* is dominated by one parent and there is no dominance relation with the other one, or when *D* is dominated by one parent and it dominates the other parent.

The “otherwise” branch refers to the case when *D* is not valid, if the user permits such an application for the operator.

In figure 1 we present the search space for a biobjective minimization problem, structured by the dominance relation between *D* and the parents P_1 and P_2, when between the parents do not manifest some dominance relation. The formula (1) is also valid if one parent dominates the other.

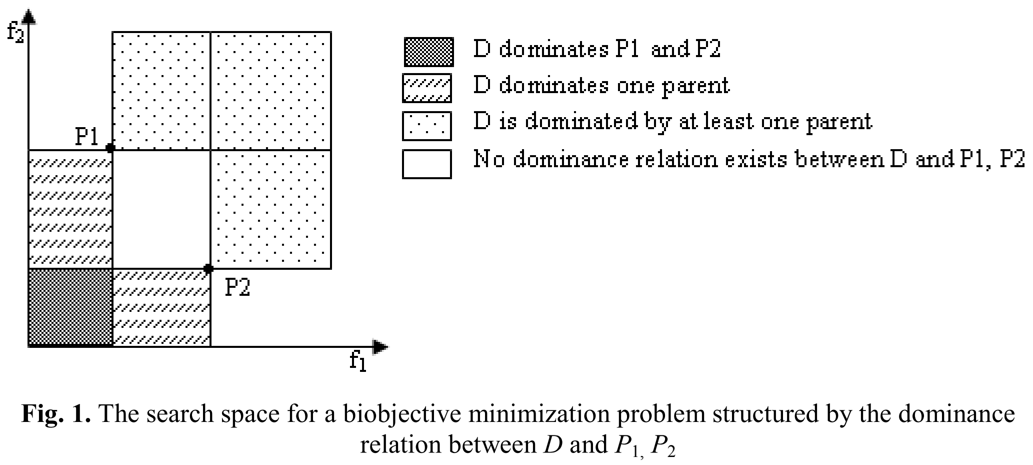
f2
f1
P1
P2
▩ D dominates P1 and P2
▨ D dominates one parent
⬚ D is dominated by at least one parent
☐ No dominance relation exists between D and P1, P2

Fig. 1. The search space for a biobjective minimization problem structured by the dominance relation between *D* and P_1, P_2

If a crossover operator produces two descendents, then the formula (1) is applied twice, for each descendent.

The range for values of parameters k_1 and k_2 is bounded by ½ and $G/2$.

During the evolution, for every application of the operator x where the correspondent descendent dominates one parent, the value $\Pi(x)$ reduces with constant step, k_1/G, until is reached the generation $G/(2k_1)$. After that, the progress becomes invariable 0.5. So, more reduced k_1, more slowly the progress converges to 0.5. Extremely, if k_1 = ½, then $\Pi(x)$ reduces with constant step until the last generation, when this will be 0.5. If $k_1 = G/2$, starting with the second generation, $\Pi(x)$ will be constantly 0.5.

The same comments are also available for the parameter k_2. During the evolution, for every application of the operator x where the correspondent descendent is dominated by at least one parent, the operator progress reduces with constant step, k_2/G, until is reached the generation $G/(2k_2)$. After that it becomes invariable 0.

If the descendents weaker than the parents are discouraged, k_2 is required to have a big value (for example $G/3$). But this will narrow the search, because the progress of x reduces during the evolution, and this operator will have smaller chances to be applied in the future.

The values of parameters k_1 and k_2 are tuned from the perspective of extent and quality of the search space exploration. For a maximum exploration and consequently a maximum diversity, both parameters are set to small values (tending to ½). It is prescribed that the k_1 value does not exceed the value of k_2, because we don't want to lose some descendents which dominates one parent and to keep some descendents dominated by at least one parent.

For the mutation operator, we modified the formula designed in [1] for progress assignment. In [1], Basseur et al. considered a progress constant (1/2) if the solution before and the solution after the operator apply do not dominate each other [1]:

$$\Pi(x) = \begin{cases} 0, \text{if } M(s) \text{ dominates } s \\ 1, \text{if } s \text{ dominates } M(s) \\ 0.5, \text{ otherwise} \end{cases}, \tag{2}$$

where $M(s)$ is the candidate-solution which results after application of x.

In this paper we assign a variable progress to the mutation operator if s and $M(s)$ can not be compared. This value depends on the parameter k_3 which enforces the convergence velocity for the progress of x to value 0.5.

So, the progress of a mutation operator x, applied to the candidate-solution s, is determined by the formula:

$$\Pi(x) = \begin{cases} 1, \text{if } M(s) \text{ dominates } s \\ 0, \text{if } s \text{ dominates } M(s) \text{ or } M(s) \text{ is not valid} \\ \max(1 - \frac{k3 * t}{G}, 0.5), \text{if no dominance relation exists between } s \text{ and } M(s) \end{cases}. \tag{3}$$

During the evolution, at every application of operator x where $M(s)$ is valid and s and $M(s)$ do not dominate each other, the value $\Pi(x)$ reduces with constant step, k_3/G, until the generation $G/(2\ k_3)$. After that, it becomes invariable 0.5. As in the case of the others parameters (k_1 and

k_2), a reduced value for k_3 allows a wider exploration of the search space. The reason is that we allow to enter in population more individuals in the same front with the individuals to whom was applied the operator.

The selection procedure for applying the genetic operators

At every crossover and mutation stage, in every generation, the operator to be applied is selected based on the selection probabilities of all the operators. For this, we compute for every operator the average progress, per application, from beginning of evolution, using the formula [1]:

$$\text{Progress}(x) = \frac{\sum_{i=1}^{||x||} \Pi_i(x)}{||x||}, \tag{4}$$

where $\Pi_i(x)$ is the progress gained by x at the i^{th} application and $\|x\|$ is the number of times when the operator x was applied.

After each operator application, its selection probability is updated, according to the formula [1]:

$$P_x = \frac{\text{Progress}(x)}{\sum_{j=1}^{n} \text{Progress}(Op_j)} * (1 - n * \delta_x) + \delta_x, \tag{5}$$

where Op_j is a generic operator in the class of x, having n elements, and $\delta_x \in (0,1)$ is the minimum value for the selection probability for every operator. This allows keeping every operator, though the weak ones.

Initially, each crossover and mutation operator has assigned the same selection probability, equal to $1/n$. For example, if we work with two crossover operators, at first generation their selection probability is 0.5. In [1], the authors used as initial selection probabilities the values $1/(n*p_{\text{mutation}})$, where p_{mutation} is the mutation ratio.

Based on the selection probabilities at last generation, we may conclude what crossover and mutation operator is the most beneficial for the considered instance.

For implementing the dynamic application of genetic operators, we use the updated selection probabilities in a roulette-wheel schema. We sort the probabilities for the crossover / mutation operators and, depending on a random value in the [0, $\sum_{j=1}^{n} \text{P}_{Op_j}$] range, we select that operator whose probability is placed in the selected region. Consequently, bigger progress of an operator, higher chances for it to be selected. This algorithm will use best operators more often.

We adopt the roulette-wheel principle because an elitist schema, where we select the operator with highest selection probability, would eliminate from the process the operators weaker that the best one.

The advantages of dynamic application of genetic operators are:

- detection for the most appropriate genetic operators to a given instance;
- promotion of the beneficial results for all the available operators;
- extension of the genetic search without loosing the direction, because different operators tend to produce different results;
- possibility to influence the search space exploration/exploitation (by k_1, k_2, k_3 values).

Although for the tests in this paper were used two crossover operators (UX, PPX) and three mutation operators (frame-shift, translocation and inversion), the presented mechanisms can be applied for any number of genetic operators.

The Population Partial Reinitialization

This mechanism was proposed as diversification mechanism, applied every time when the premature convergence risk attains a level considered critical. This condition is satisfied if, at the current generation, the average progress of every operator is smaller than a minimum threshold, *pMin* $\in [0,1]$, set before the evolution starts. The *pMin* value is set in relation with δ_x values.

The reinitialization consists in replacing a part of the new population with random generated individuals, in a proportion set before the algorithm run (*pReinit*).

In addition, if the reinitialization is performed 300 times in a run, the evolution will be stopped, though the stop criteria, initially set, was not satisfied. The reason is that a big number of reinitialization signifies minimal chances to identify better solutions than current ones.

Implementation of the Two Mechanisms in the NSGA_II Algorithm

These two mechanisms were implemented in the algorithm NSGA_II [5], one of the best genetic algorithms designed for complex problems. The new algorithm was named NSGA_II DAR (*Dynamic Application of genetic operators and population partial Reinitialization*).

In order to validate the proposed mechanisms, we compared the results obtained by NSGA_II and the NSGA_II DAR algorithms for a known difficult test instance in the class JSSP (Job Shop Scheduling Problem).

The NSGA_II (*Non-dominated Sorting Genetic Algorithm*), designed by Deb et al. in 2000 for multiobjective difficult problems, sorts the population according to the level of non-domination, each solution being compared with every other solution to find if it is dominated. This way there are constructed one by one the non-dominated fronts (*F*), each consisting in individuals non-dominated by those in the subsequent fronts. NSGA_II uses as diversity preservation mechanism a crowding distance comparison operator, which guides the selection process towards the true Pareto-optimal front, by favoring the solutions in less dense regions in every front. This algorithm uses a binary tournament selection, where the selection criterion is based on this operator.

The pseudocode of the NSGA_II DAR algorithm is the following:

```
1. $t \leftarrow 0$                                   first generation
2. pseudo-random initialization of the initial
population, $P_t$
3. quick_sort($P_t$)
4. $Q_t \leftarrow$ new_population($P_t$)            dynamic application of operators => new population $Q_t$
5. while evolution is not ended
  5.1. $R_t \leftarrow P_t \cup Q_t$                parent population combines with children population
  5.2. $F \leftarrow$ quick_sort($R_t$)              $F = \{F_0, F_1, ...\}$ are the fronts of population $R_t$
  5.3. $P_{t+1} \leftarrow \varnothing$ and $i \leftarrow 0$     initialization of the next parent population
  5.4. until $|P_{t+1}| + |F_i| \le N$               until the parent population is complete
    5.4.1. $P_{t+1} \leftarrow P_{t+1} \cup F_i$     the front $F_i$ is included in parent population
    5.4.2. crowd_dist($F_i$)                          compute the crowding distances in $F_i$
    5.4.3. $i \leftarrow i+1$                         next front
    5.5. sort($F_i$, $\geq_n$ )                       descending sort based on relation $\geq_n$ (the crowding
                                                      distance comparison operator)
    5.6. $P_{t+1} \leftarrow P_{t+1} \cup F_i[1:(N-|P_{t+1}|)]$    if the current front contains much more individuals
                                                      than necessary for completion the population ($k$), we
                                                      include only the first $k$ solutions in these
    5.7. $Q_{t+1} \leftarrow$ new_population($P_{t+1}$)     dynamic application of operators => new population
                                                      $Q_{t+1}$
    5.8. if reinit_criterion then                     if the reinitialization criterion is satisfied, the
           population partial reinitialization        population partial reinitialization is performed
    5.9. $t \leftarrow t+1$                           next generation
6. return the best solutions
```

The final solutions for the algorithms are all the individuals, structurally different, having the best objective value.

Simulation Results

In this study, the mechanisms, implemented in the NSGA_II DAR algorithm, were tested on the ft10 test instance, one of the most difficult JSSP.

JSSP is formally defined by the following [6]:

Input data:

- a set M of $m \in Z^+$ resources (machines);
- a set J of jobs, each job $i \in J$ consisting in a sequence of n_i operations $o_{i,j}$ with $1 \leq j \leq n_i$;
- for every operation we know the machine which processes it ($m_{i,j}$) and the processing time ($\tau_{i,j} \in N$, $i \in J$, $1 \leq j \leq n_i$).

Requirement:

A schedule for J - a collection of machine schedules $f_m : \{o_{i,j} \mid m_{i,j} = m\} \rightarrow N$ such that:

- $f_m(o_{i,j}) > f_m(o_{i',j'})$ implies $f_m(o_{i,j}) \geq f_m(o_{i',j'}) + \tau_{i',j'}$ (the operation $o_{i,j}$ once started, the machine which processes it will become available only after $o_{i,j}$ is finished) and
- $f_m(o_{i,j+1}) \geq f_m(o_{i,j}) + \tau_{i,j}$ (for every operation, its successor in the same job will be processed only after it is finished).

Here are stated the cumulative constraints of JSSP: non-preemption constraint, precedence constraint and capacity constraint (a machine processes one operation at a time). The last one is understood from input data.

Objective: minimization of makespan for the schedule, namely:

$$C^*_{\max} = \min(C_{\max}) . \tag{6}$$

The formula for the makespan implies the detailed description of the genetic encoding for the JSSP; the goal of our study being the comparative analysis for the performance of the solutions (the makespan), we choose to not present also the structure of the solutions; this is though available upon request.

The experience shows that JSSP is not only NP-difficult, but very difficult to solve even heuristically [4]. Under these circumstances, avoiding the premature convergence of the algorithm to suboptimal regions for the JSSP is a basic requirement.

In the ft10 test-instance [2], 10 jobs, each formed by 10 operations, must be scheduled on 10 machines. The candidate solutions are formed only by 100 genes, but its complexity level is pretty high [2]. It is one of the most difficult JSS problems. The best solution known has the makespan 930.

The algorithm was run 5 times for each set of parameters values, presented in the table 1.

Table 1. Parameters values used for the simulation

	Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Test 6**
G	100	200	200	400	500	500
Mutation rate	0.05	0.05	0.01	0.01	0.05	0.03
k_1	4	4	4	100	100	150
k_2	10	10	10	200	200	220
k_3	5	5	5	100	100	200
δ_i, $i \in$ {UX, PPX, frame-shift, translocation, inversion}	0.5,0.8,0.05 0.01,0.01	0.5,0.8,0.05 0.01,0.01	0.5,0.8, 0.05,0.01, 0.01	0.5,0.8, 0.05,0.01, 0.01	0.2,0.3, 0.05,0.05, 0.05	0.2,0.3, 0.05, 0.05,0.05
pMin	0.5	0.3	0.3	0.4	0.35	0.4
pReinit	10%	10%	10%	20%	20%	50%

The solutions quality is interpreted from many points of view: the best performance (C^*_{max}) obtained in the 30 tests, the average performance, the worst performance, variation range for C^*_{max} and the run time per generation.

In order to evaluate the usefulness of the new mechanisms, we compare the results obtained by the new algorithm, NSGA_II DAR, and three others genetic algorithms: the canonic genetic algorithm, an elitist genetic algorithm and the NSGA_II in the absence of these mechanisms.

The comparative study (see table 2) shows that the NSGA_II DAR algorithm is the most efficient taking into account the first three performance measures. It obtained the solution with the best makespan - 1013.

Table 2. Performance measures for the ft10 test-instance

Algoritm / **Measure**	**Canonic**	**Elitist**	**NSGA_II**	**NSGA_II DAR**
Best performance, C^*_{max}	1342	1054	1216	**1013**
Average performance	1384	1213	1265	**1102**
Worst performance	1553	1355	1345	**1306**
Variation range for C^*_{max}	211	301	**129**	293
Run time, per generation, sec.	**0.11**	3.3	1.75	0.7

The variation range for the makespan, which is another measure for the diversity of the solutions, is though better for the NSGA_II. The best run time is obtained for the canonic genetic algorithm, the simplest one, but this information has value only if we strongly need a quick run and we accept for this poor quality solutions.

Conclusions

In the paper we submitted to research two mechanisms to avoid the premature convergence of genetic algorithms. These are the dynamical application of crossover and mutation operators and the population partial reinitialization. The numerous tests run to evaluate their quality prove the beneficial role of both mechanisms in diversification the population.

The dynamical application of genetic operators achieves slowly this goal, during all the evolution process, simultaneously with producing supplementary advantages: the identification of the operators adequate to the instance and the promoting the beneficial effects of all the available operators.

The population partial reinitialization achieves this goal only in the critical moments, when the risk of premature convergence is big enough. Hence, on a problem not so difficult, if this risk does not occur, the individuals are not replaced by some randomly generated new individuals, not being necessary.

The improvement of genetic algorithms by the two mechanisms is validated by the results of a comparative analysis on the ft10 JSSP problem; the new algorithm (the NSGA_II with the two mechanisms, NSGA_II DAR) is set against with three others genetic algorithms: the canonic genetic algorithm, an elitist genetic algorithm and the NSGA_II in the absence of the mechanisms. Viewed by the most important perspectives - the best performance obtained in 30 tests, the average performance and the worst performance of the solutions – the new algorithm is the most efficient. Only according to the run time per generation and to the variation range for the makespan the other algorithms are better than the new algorithm. As a final conclusion, the proposed algorithm, NSGA_II DAR, is able to obtain for difficult problems better results than all the others compared algorithms.

These results induce the idea of a mutual beneficial influence of the genetic operators in the NSGA_II DAR.

Many further developments on this topic are possible. We may use many more genetic operators, we may slightly adapt the formulas for the progress of the operators to the instance to be solved or we may vary the values of the interdependent parameters.

References

1. Basseur, K.M., Seynhaeve, F., Talbi, E. - Design of multi-objective evolutionary algorithms: application to the flow-shop scheduling problem, *Proceedings of the 2002 Congress on Evolutionary Computation* (CEC), Honolulu, Hawaii, IEEE Press, vol. 2, pp. 1151–1156, 2002
2. Beasley, J.E. - OR library: distributing test problems by electronic mail, *European Journal of Operational Research* 41, pp. 1069-1072, 1990
3. Belea, R., Caraman S., Palade, V. - Diagnosing the Population State in a Genetic Algorithm Using Hamming Distance, *the Proceedings of Knowledge-based and Intelligent Engineering Systems* KES 2004, pp. 246-255, 2004
4. Brucker, P., Knust, S. - *Complexity results for scheduling problems*, Osnabruck University, 2005
5. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T. - A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II, *IEEE Transactions on Evolutionary Computation*, 6(2), pp.182–197, 2002
6. * * * , *A compendium of NP optimization problems, in Complexity and approximation combinatorial optimization problems and their approximability properties*, by Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M., Springer Verlag, 2004

Mecanisme pentru evitarea convergenței premature a algoritmilor genetici

Rezumat

Optimizarea realizată cu algoritmi genetici este însoțită adesea de convergența prematură a algoritmului, în special în cazul problemelor multimodale. În lucrarea de față propunem şi testăm două mecanisme pentru evitarea convergenței premature a algoritmilor genetici prin menținerea diversității populației de-a lungul evoluției, în două moduri diferite. Aceste mecanisme sunt aplicarea dinamică a mai multor operatori genetici, bazată pe progresul mediu, şi reinițializarea parțială a populației. Mecanismele au fost testate prin implementarea in algoritmul NSGA_II, care a fost apoi aplicat uneia dintre cele mai dificile probleme-test de tip JSSP (Job Shop Scheduling Problems), ft10. Analiza comparativă a noului algoritm cu algoritmul NSGA_II în absența mecanismelor propuse, ca de altfel şi cu un algoritm genetic elitist şi cu cel canonic, demonstrează utilitatea ambelor mecanisme propuse.