ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1996
Seria: AUTOMATYKA z. 117 Nr kol. 1337

Adam JANIAK, Krzysztof CHUDZIK, Marie-Claude PORTMANN
Politechnika Wrocławska; Ecole des Mines de Nancy

COMPARISON OF FOUR GENETIC ALGORITHMS ON THE BASIS OF SOME FLOW-SHOP PROBLEM

Summary. The paper is devoted to the comparison of four genetic algorithms on the basis of some flow-shop problem. It was assumed that models of job processing times in this problem are convex functions of resources. In some of the constructed algorithms there were used specific cross-over and mutation techniques which employed the specific problem properties. There are given several experimental results with proposed genetic algorithms together with their analysis.

PORÓWNANIE CZTERECH ALGORYTMÓW GENETYCZNYCH NA BAZIE SEKWENCYJNEGO PROBLEMU PRZEPŁYWOWEGO

Streszczenie. Praca jest poświęcona porównaniu czterech algorytmów genetycznych na bazie problemu przepływowego. Przyjęto założenie, że modele czasów trwania wykonywania zadań są wypukłymi funkcjami zasobów. W części skonstruowanych algorytmów były użyte specyficzne techniki krzyżowania i mutacji, które wykorzystywały specyficzne własności problemu. Podane zostały wyniki eksperymentów numerycznych z zaproponowanymi algorytmami genetycznymi razem z ich analizą.

1. Introduction. Problem formulation

The paper deals with the extension of a classical permutation flow-shop problem to the case when processing times of jobs are not given, but they are convex decreasing functions of locally and globally constrained resources. This kind of problems appears in many manufacturing processes. In [1] it was proved that the problem under consideration is NP-hard even for the two machine case. The branch and bound algorithm for this problem was constructed in [1]. It was able to solve optimaly problems with only about 100 operations. In this paper we construct four genetic algorithms to solve it. The first (not fully successful) attempts at using genetic approach in order to solve flow shop problem with some resource constraints were made in [2,3] for linear models.

The problem may be formulated as follows.

There are n jobs $J_1, J_2, \ldots, J_n$ (available for processing at time zero) which have to be processed on m (continuously available) machines $M_1, \ldots, M_m$ in that order. The processing orders of jobs on all machines are the same. Each machine M_ν, $\nu = 1,2,\ldots,m$, can handle at most one job at a time. Job J_j, $j = 1,2,\ldots,n$, thus consists of a sequence of m operations $O_{j1}, O_{j2}, \ldots, O_{j\nu}, \ldots, O_{jm}$, where $O_{j\nu}$ corresponds to the processing of job J_j on machine M_ν during an uninterrupted processing time $p_{j\nu}$. It is assumed that the processing time $p_{j\nu}$ of the operation $O_{j\nu}$ depends on an amount of resource $u_{j\nu}$, allotted to perform this operation in the following way:

$$p_{j\nu} \stackrel{\Delta}{=} f_{j\nu}(u_{j\nu}), \quad j = 1,2,\ldots,n, \ \nu = 1,2,\ldots,m,$$

where $f_{j\nu}(\cdot)$ are convex decreasing functions.

To avoid complicated notation, in the sequel it will be assumed that processing times of all the operations are dependent on a common resource, however, the results obtained in the paper are also valid for the cases when the separate machines demand different resources for realization of operations (and the other do not require any additional resources – see the tested examples in Section 3).

It will be assumed that a resource allocation $r \stackrel{\Delta}{=} [r_1, r_2, \ldots, r_\nu, \ldots, r_m]$ (where $r_\nu \stackrel{\Delta}{=} [r_{1\nu}, \ldots, r_{i\nu}, \ldots, r_{n\nu}]$) is feasible if the following constraints are satisfied:

$$\alpha_{j\nu} \le r_{j\nu} \le \beta_{j\nu}, \quad j = 1,2,\ldots,n; \ \nu = 1,2,\ldots,m,$$

$$\sum_{j=1}^{n} \sum_{\nu=1}^{m} r_{j\nu} \le \hat{R},$$

where $\alpha_{j\nu}$, $\beta_{j\nu}$ are, respectively, the lower and the upper limits on the amount of resource allotted to a realization of the operation $O_{j\nu}$, $f_{j\nu}(\alpha_{j\nu})$ and $f_{j\nu}(\beta_{j\nu})$ ($f_{j\nu}(\beta_{j\nu}) > 0$) are the maximum (normal) and minimum (crash) durations of $O_{j\nu}$, $0 \le \alpha_{j\nu} \le \beta_{j\nu} \le \hat{R}$, and $\hat{R}$ is the global amount of resource allotted to a realization of all the jobs, $\hat{R} \ge \sum_{j=1}^{n} \sum_{\nu=1}^{m} \alpha_{j\nu}$.

The set of all the feasible resource allocations will be denoted by R.

A processing order of jobs will be represented by a permutation $\pi=(\pi(1),\pi(2),\ldots,\pi(j),\ldots,\pi(n))$ of the job indices $1,2,\ldots,n$, where $\pi(j)$ denotes that index of a job which is in the position j of π. Let Π denote the set of all such permutations.

Moreover, for a permutation $\pi\in\Pi$ and a resource allocation $r\in R$ by $C_{\max}(\pi,r)\triangleq\max_{1\le j\le n}\{C_{\pi(j)}(\pi,r)\}$ will be denoted the maximum job completion time, where $C_{\pi(j)}(\pi,r)$ denotes the completion time of job $J_{\pi(j)}$ (precisely, the operation $O_{\pi(j)m}$).

The problem is to find such a control, i.e. such a job processing order $\pi^*\in\Pi$ and such a feasible resource allocation $r^*\in R$ that the maximum job completion time is minimized.

2. Problem properties

By r_π^* will be denoted the optimal resource allocation for a permutation π, i.e. a resource allocation for which $C_{\max}(\pi,r_\pi^*)=\min_{r\in R}C_{\max}(\pi,r)$. The optimum resource allocation $r_\pi^*\in R$ for each $\pi\in\Pi$ may be obtained by the known algorithms of the convex programming and for the case with linear models of operations is obtained by applying the algorithm of Hamacher and Tufekci with some modifications (see [1] for description).

It is easy to notice that the following equation holds:

$$C_{\max}(\pi,r_\pi^*)=\max_{1\le j_1\le\ldots\le j_v\le\ldots\le j_{m-1}\le n}\left[\sum_{k=1}^{j_1}f_{\pi(k)1}\left(r^*_{\pi(k)1}\right)+\sum_{v=2}^{m-1}\sum_{k=j_{v-1}}^{j_v}f_{\pi(k)v}\left(r^*_{\pi(k)v}\right)+\sum_{k=j_{m-1}}^{n}f_{\pi(k)m}\left(r^*_{\pi(k)m}\right)\right]. \quad (1)$$

This equation follows from the fact, that the maximum job completion time $C_{\max}(\pi,r_\pi^*)$ is equal to the duration time between the beginning moment of the realization of the first job $J_{\pi(1)}$ in π on the first machine and the completion moment of the last job $J_{\pi(n)}$ on the last machine. It is easy to notice that this duration time, in turn, is equal to the sum of the processing times of operations from operation sequence of the form (1) for which realization of each operation starts immediately after the completion of the preceding operation from this sequence. In other words, there is no inserted idle time between the realization of the succeeding operations of this operation sequence. This kind of operation sequence is called critical path and positions $j_1\le j_2,\ldots,\le j_v\le,\ldots,\le j_{m-1}$ for which

$$C_{\max}\left(\pi, r_{\pi}^{*}\right)=\sum_{k=1}^{j_1} f_{\pi(k)1}\left(r_{\pi(k)1}^{*}\right)+\sum_{v=2}^{m-1}\sum_{k=j_{v-1}}^{j_v} f_{\pi(k)v}\left(r_{\pi(k)v}^{*}\right)+\sum_{k=j_{m-1}}^{n} f_{\pi(k)m}\left(r_{\pi(k)m}^{*}\right) \qquad (2)$$

are called the critical path positions.

A subsequence of the consecutive jobs from the permutation $\pi \in \Pi$ between succeeding (different) critical path positions of all the critical paths in $\pi \in \Pi$ (under $r_{\pi}^{*} \in R$) is called a section (see [1] for precise definition).

It was proved in [1] that:

Property 1

It is not possible to obtain better solution (i.e. a permutation with the smaller value of $C_{\max}$) by the interchanging the jobs inside each section. The improvement of $C_{\max}$ is possible only by moving the jobs outside of the sections.

The precise formulation of this property is in [1]. This property will be applied in some genetic algorithms in the next section.

3. Outline of the genetic algorithms

There were constructed four different genetic algorithms to solve the problem under consideration. The general scheme [4] of them is as follows:

$N := 0$,

A) Initial Phase,

B) Survival Phase,

C) Cross-over Phase,

$N := N + 1$,

D) Mutation Phase,

E) If (no Stop Test) then go to B.

In initial phase of all the algorithms the initial populations were obtained by approximate algorithms of Dannenbring, Campbell, Dudek, Smith and Nawaz which were generalized [1] to the case with resource.

In cross-over phases were used classical cross-over technique [4] or specific cross-over technique [3] in which Property 1 was employed. The mutation phase was constructed in the similar way.

Finally we considered the following four implementations of genetic algorithm (GA):

GA1 – G.A. with specific cross-over and with specific mutation;

GA2 – G.A. with classical cross-over and with specific mutation;

GA3 – G.A. with specific cross-over and with classical mutation;

GA4 – G.A. with classical cross-over and with classical mutation.

The examples on which the above mentioned genetic algorithms were tested were described in [3].

4. Presentation of results

4.1. Comparison of quality of solutions

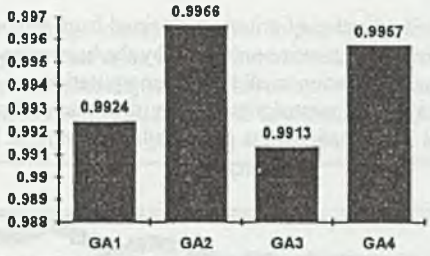

Fig.1. Average value of the best value of criterion function obtained in all time of computation normalised by the best value of criterion in initial population

Rys.1. Średnia wartość najlepszej wartości funkcji kryterialnej otrzymana w całym czasie obliczeń normalizowana przez najlepszą wartość kryterium w populacji inicjującej

The above results show that the algorithm with the best efficiency of criterion value improvement is GA3, i.e. the one with specific cross-over and classical mutation.

Fig. 1 and 2 presented the ratios obtained after considered iterations (N = 0, ..., 50) from the beginning. Now in Fig. 3, 4, 5 and 6 some criterion value ratios calculated in separate iterations will be presented.

Fig. 3 shows that the algorithms GA2 and GA4 lost the best solutions, whereas the algorithms GA1 and GA2 improve them. At the same time the worst solutions were preserved better by the algorithms GA1 and GA2 (see Fig. 5) and the algorithms GA3 and GA4 reduced them the best.

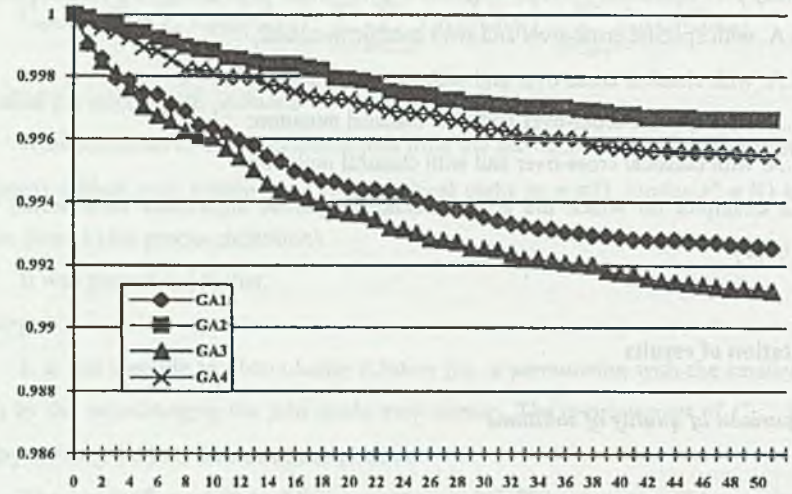

Fig.2. Average value of the best value of criterion obtained from the beginning of computation to iteration which is shown on x-axis normalised by the best value of criterion in initial population in all time of computation

Rys.2. Średnia wartość najlepszej wartości kryterium otrzymana od rozpoczęcia obliczeń do iteracji pokazanej na osi X normalizowana przez najlepszą wartość kryterium w populacji inicjującej

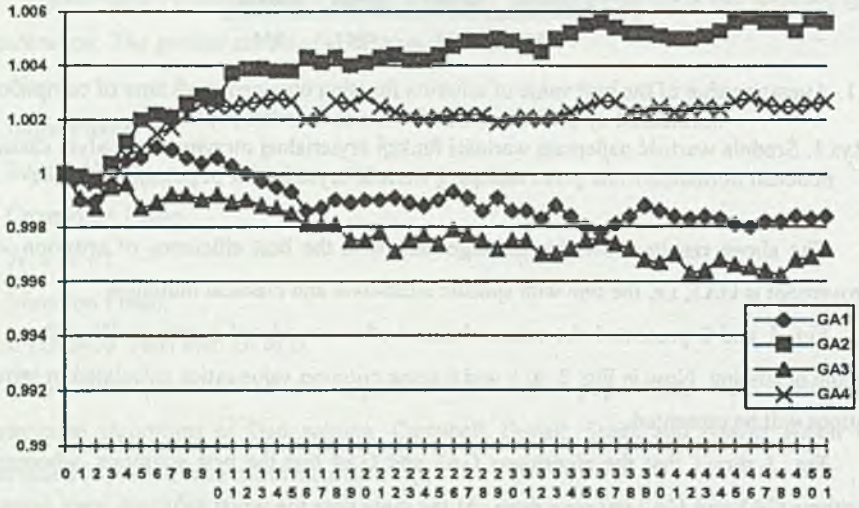

Fig.3. Average value of the best value of criterion in population in each iteration normalised by the best value of criterion in initial population

Rys.3. Średnia wartość najlepszej wartości kryterium w populacji w każdej iteracji normalizowana przez najlepszą wartość kryterium w populacji inicjującej

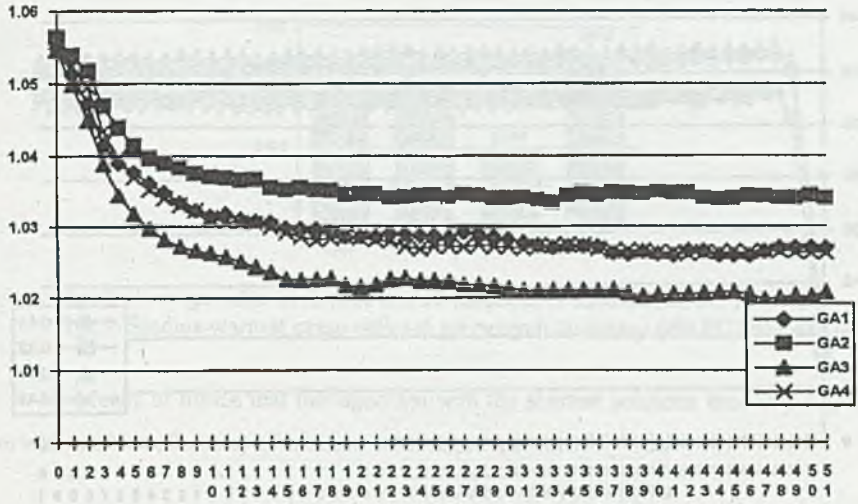

Fig.4. Average value of average value of criterion in population in each iteration normalised by the best value of criterion in initial population

Rys.4. Średnia wartość średniej wartości kryterium w populacji w każdej iteracji normalizowana przez najlepszą wartość kryterium w populacji inicjującej

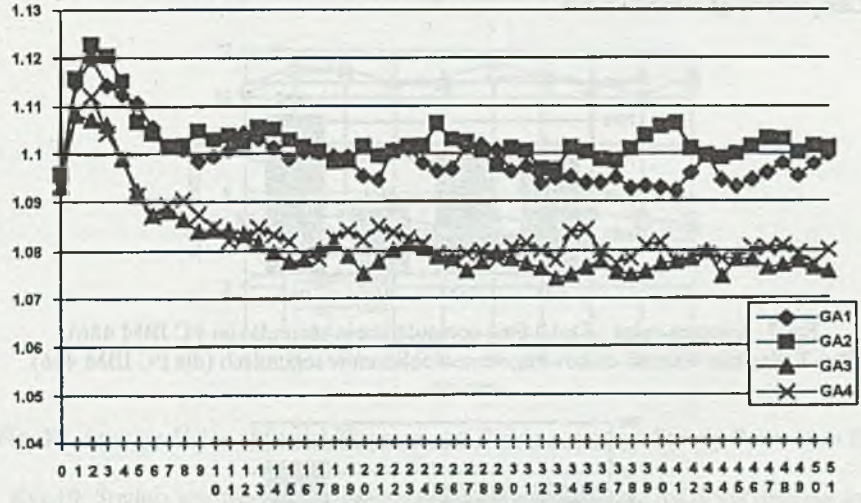

Fig.5. Average value of the worst value of criterion in population in each iteration normalised by the best value of criterion in initial population

Rys.5. Wartość średnia najgorszej wartości kryterium w populacji w każdej iteracji normalizowana przez najlepszą wartość kryterium w populacji inicjującej

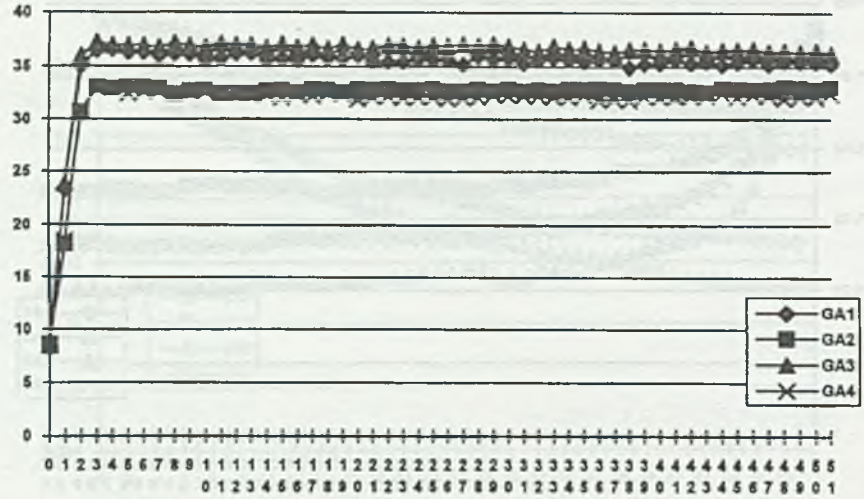

Fig.6. Average value of size of population in iteration
Rys.6. Średnia wartość liczności populacji w danej iteracji

4.2. Comparison of solution times

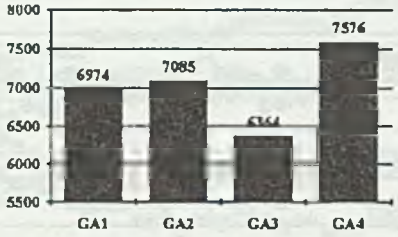

Fig.7. Average value of total time computation in seconds (on PC IBM 486)
Rys.7. Średnia wartość całkowitego czasu obliczeń w sekundach (dla PC IBM 486)

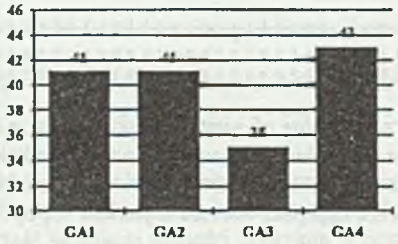

Fig.8. Average value of time of initial phase in seconds (on PC IBM 486)
Rys.8. Średnia wartość czasu fazy inicjacji w sekundach (dla PC IBM 486)

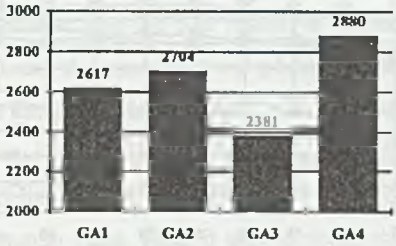

Fig. 9. Average value of time of first 20 iterations in seconds (on PC IBM 486).
Rys.9. Średnia wartość czasu obliczeń pierwszych 20 iteracji (dla PC IBM 486)

It is easy to notice that the algorithm with the shortest solutions time is GA3, i.e. the one with the best efficiency of criterion value improvement.

4.3. Comparison of cross-over operators

The results of comparison are given in Fig. 10, 11, 12 and 13. ("Partial" means that results were calculated for each five succeeding iteration of algorithm.)

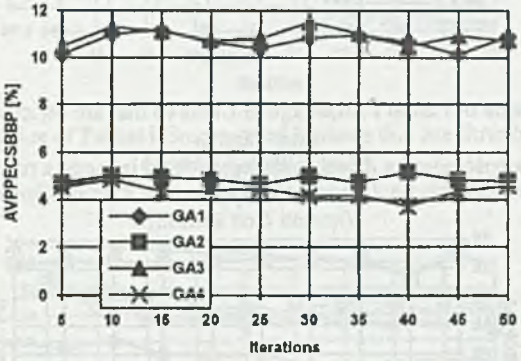

Fig. 10. Average Value of Partial PErcentage of Children that are Strictly Better than Both Parents
Rys.10. Średnia wartość odsetka dzieci ściśle lepszych od obojga rodziców (liczona co 5 iteracji)

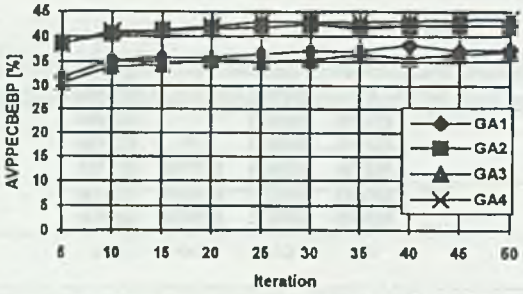

Fig. 11. Average Value of Partial PErcentage of Children that are Better or Equivalent to Both Parents

Rys.11. Średnia wartość odsetka dzieci lepszych lub równoważnych obojgu rodzicom (liczona co 5 iteracji)

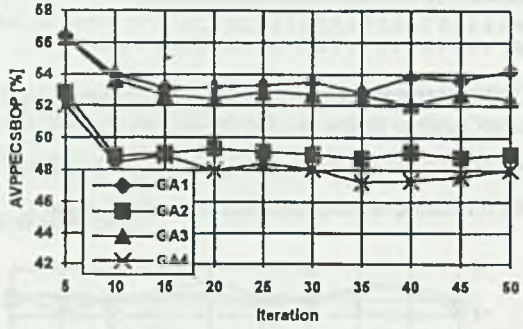

Fig.12. Average Value of Partial PErcentage of Children that are Strictly Better than One of their Parents

Rys.12. Średnia wartość odsetka dzieci ściśle lepszych od jednego z rodziców (liczona co 5 iteracji)

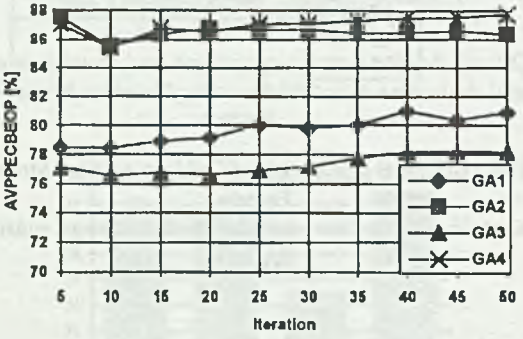

Fig.13. Average Value of Partial PErcentage of Children that are Better or Equivalent to One of their Parents

Rys.13. Średnia wartość odsetka dzieci lepszych lub równoważnych jednemu z rodziców (liczona co 5 iteracji)

It is easy to notice in Fig. 10 and 12 that the algorithms GA1 and GA3, i.e. the algorithms with specific cross-over give the best results. It means that the best cross-over operator is the specific one. Using this operator it was possible to obtain 10% children strictly better than both parents. It is also interesting that the classical operator obtains the percentage of children that are better or equivalent to one or both parents, which is better than the specific one. It follows from the above that the classical cross-over operator generates a lot of solutions which do not change the criterion value. Thus, it easy to notice that a utilisation of specific problem properties in cross-over operator can improve the algorithm work.

4.4. Comparison of mutation operators

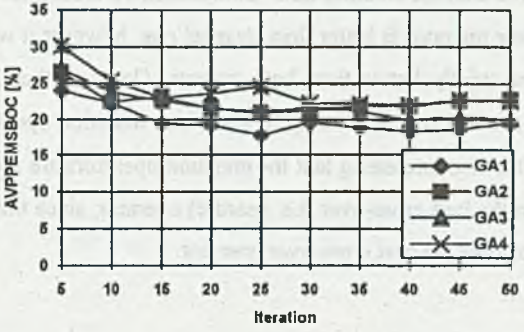

Fig. 14. Average Value of Partial PErcentage of Mutants that are Strictly Better than the Original Chromosome

Rys.14. Średnia wartość odsetka mutantów ściśle lepszych od oryginalnego chromosomu (liczona co 5 iteracji)

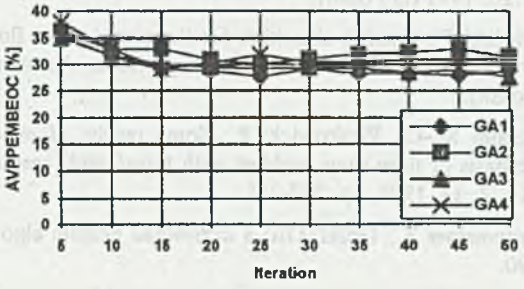

Fig. 15. Average Value of Partial PEercentage of Mutants that are Better or Equivalent to the Original Chromosome

Rys.15. Średnia wartość odsetka mutantów lepszych lub równoważnych jednemu z rodziców (liczona co 5 iteracji)

Analysing Fig. 14 and 15 it is not easy to say which of mutation operators is better since the results are similar for all of them. However it should be stressed on the fact that the mutation operators are more efficient than the best (i.e. specific) cross-over operator since the last one can obtain about 10% solutions better than their parents and the mutation operators can obtain about 20% solutions better than their predecessor.

5. Conclusion remarks

It follows from the above comparison of four genetic algorithms, that the algorithm GA3 with specific (i.e. employing some problem elimination properties) cross-over and the classical mutation operator has the best efficiency of criterion value improvement. This algorithm has also the shortest solution time. Comparison of cross-over operators shows that the specific cross-over operator is better than classical one, however it was able to obtain only about 10% children strictly better than both parents. On the other hand, comparison of mutation operators show that it is not easy to say which mutation operator is better since the results are similar. It is also interesting that the mutation operators are more efficient in finding better solutions than the best cross-over (i.e. specific) operator, since they can find about 10% more better solutions than the best cross-over operator.

REFERENCE

1. Janiak A.: Exact and Approximate Algorithms of Job Sequencing and Resource Allocation in Discrete Manufacturing Processes. Monograph published by Technical University of Wrocław, No 87/20, 1991 (in Polish).
2. Janiak A., Kobylański P.: Genetic algorithm for the permutation flow-shop problem with resource. Zeszyty Naukowe Politechniki Śląskiej, Seria Automatyka, z. 114, 1994, pp. 99-109 (in Polish).
3. Janiak A., Portmann M.-C., Płaskowicki P.: Some results of experiments with genetic algorithm on the basis of flow-shop problem with transferable resources. Elektrotechnika, Wyd. AGH, T. 14, Z. 13, 1995, pp. 247-255.
4. Whitley D., Starkweather T.: GENITOII: a distributed genetic algorithm. J. Expt. Theor. Artif. Intell., 1990.

Recenzent: Dr hab. inż Konrad Wala, prof. AGH

Wpłynęło do Redakcji do 30.06.1996 r.

Abstract

In the paper we make a comparison of four genetic algorithms on the basis of flow type manufacturing problem. It was assumed that models of job processing times in this problem are not given (constant) but they are convex functions of some limited recources. There were considered two different cross-over and mutation phases. One cross-over and one mutation phase was classical one and other cross-over and mutation phase was specific one with employed some specific elimination properties of the problem. The combination of these phases yield the considered four genetic algorithms. There are given several experimental results with proposed genetic algorithms together with their analysis. It follows from this analysis that the best algorithm is one with specific cross-over phase and classical mutation one. It is also interesting that the mutation operators are more efficient in finding better solutions than the best cross-over (i.e. specific) operator since they can find about 10% more better solutions than the best cross-over operator.