Int J Adv Manuf Technol (2011) 57:735–752
DOI 10.1007/s00170-011-3297-3

ORIGINAL ARTICLE

Solving the no-wait job-shop problem by using genetic algorithm with automatic adjustment

Wojciech Bożejko · Mariusz Makuchowski

Received: 11 May 2010 / Accepted: 23 March 2011 / Published online: 21 April 2011

Abstract This paper describes a methodology of automatic genetic algorithm parameters adjustment dedicated to a job-shop problem with a no-wait constraint with a makespan criterion. The numerical results show that in a given problem, the efficiency of an algorithm with auto-tuning is placed at the level of an algorithm steered in a classical way with the best-fit steering parameters.

Keywords Scheduling · Job-shop problem · No-wait constraint · Genetic algorithm

Mathematics Subject Classifications (2010) 90B36 · 90C35

1 Introduction

Every step we take in our surrounding environment proves to us that we can encounter different kinds of problems of discrete optimization. To this group, we can count not only the choice and means of transport to work but also the arrangement of tools in the cupboard. The ability of generating good solutions in the meaning of a chosen criterion (for instance the time of journey, its cost, or the accessibility of tools in the cupboard) definitely facilitates our lives. In the examples presented, generating a good solution is principally a question of any intuition. However, it is different in bigger undertakings (for instance, in planning the pipeline network in a big city or choosing a schedule for the mass production of some units) where generating a good solution is not an easy job. Moreover, applying a better solution (than the one applied so far) will bring visible profits (for instance, in the use of raw materials or shortening the time of production). To find a solution to such kinds of problems, different types of advanced algorithms are applied. Some part of algorithms dedicated to the very narrow group of problems do not have their analogies in other situations. Nevertheless, there are algorithms being applied to a wide variety of problems based on a common idea. Examples of general schemes of acting include: the exact method—branch and bound method, approximate methods like tabu search method, simulated annealing method, or genetic algorithm method.

These schemes do not specify in what way the solution must be found, how to implement each separate elements of algorithm, or how to select the steering parameters to a given problem; however, they define the overall draft of an algorithm. At the same time, it is widely known that scientists are eager to find an algorithm that would adapt its parameters to a given problem. This work attempts to create a universal genetic algorithm that would automatically select all the crucial parameters based on a statistical information instance of a given problem. Auto-tuning presented here will take place not only in a single process off-line (for a given set of examples) in an initiating phase of an algorithm but also in on-line—during its work. We will

W. Bożejko (✉) · M. Makuchowski
Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland
e-mail: wojciech.bozejko@pwr.wroc.pl

M. Makuchowski
e-mail: mariusz.makuchowski@pwr.wroc.pl

verify the quality of auto-tuning procedures through a wide variety of numerical examinations. In this case, we will assess the efficiency of genetic algorithm with auto-tuning methods applied to an analogical algorithm manually manipulated (in an optimal way). Moreover, we will carry out a test comparing the genetic algorithm equipped with all auto-tuning techniques to a known literature algorithm.

2 Description of the problem

The tested algorithms are dedicated to a job-shop problem with an additional no-wait constraint. The makespan of all jobs is assumed as a criterion of optimization. The algorithm presented here is a three-field Graham's notation (Graham et al. [4]) which is denoted as $J|\mathrm{no}-\mathrm{wait}|C_{\max}$. This problem varies from its classic equivalent (through a classic equivalent, the job-shop problem with no constraints is understood) where the requirement of the beginning time of an operation should be exactly as that time when the technological predecessor finishes its operation execution. This constraint is very often met not only in those branches of industry where the transformed product changes quickly its physical–chemical properties, e.g., in drug production (Raaymakers and Hoogeveen [9]), foodstuff production (Hall and Sriskandarajah [6]), steel melting (Wismer [16]), or concrete elements manufacturing (Grabowski and Pempera [5]), but also in other branches like in semiconductors testing (Ovacik and Uzsoy [14]) or in computer systems (Reddi and Ramamoorth [10]).

Solving methods was proposed by several authors. Schuster and Framinan [2, 12] proposed a local search-based approximative procedures for no-wait job-shop scheduling. Schuster [11] described a tabu search algorithm and considered complexity of subproblems of the no-wait job shop. Bożejko and Makuchowski [1] proposed an efficient hybrid tabu search algorithm for the considered problem.

2.1 Mathematical model

There is a given set of n jobs $J = \{1, 2, \ldots, n\}$ which are executed on a set of machines $M = \{1, 2, \ldots, m\}$. Additionally, for each job ($k \in J$) there is a sequence $O_k = (o_k^1, o_k^2, \ldots, o_k^{r_k})$, including r_k operations. The amount of all operations in this process is denoted by $o = \sum_{k\in J} r_k$. The operation $o_k^l \in O_k,\ \ l \in \{1, 2, \ldots, r_k\},\ \ k \in J$ consists of a pair (m_k^l, p_k^l) denoted as follows: first, m_k^l—the used machine and second, p_k^l—the lasting time of the operation. Moreover, in the presented model there exist three types of constraints:

1. *Sequencing*: the operation of k job must be executed in O_k sequence,
2. *Synchronic*: at the particular moment, each machine can execute not more than one operation and more than one operation of a given job cannot be carried out at the particular moment,
3. *No-wait*: each operation excluding the first one of a given job has to begin exactly at the moment of finishing the previous operation's conduction of the same job.

The solution to the classic job-shop problem defined as a set of moments ($S_k^l \geq 0$) of the beginning of o_k operation was executed. However, it must clearly be seen that in case of an additional no-wait constraint, there is always a condition

$$S_k^{l+1} = S_k^l + p_k^l, \quad l \in \left(1, 2, \ldots, r_k - 1\right).$$

From the above equation, we can infer directly that unambiguous setting of the beginning time of some jobs might be done on the basis of the beginning moment of other operation of the same job. Let S_k denotes the beginning time of the k from a definition equating the beginning moment of the first job operation, $S_k = S_k^1$. As a solution to a job-shop problem with no-wait constraint, we take a vector of non-negative terms of jobs' beginning

$$S = \left(S_1, S_2, \ldots, S_n\right).$$

On the basis of S vector, the beginning moment of S_k^l of operation execution o_k^l can be defined from the formula:

$$S_k^l = S_k + \sum_{i=1}^{l-1} p_k^i.$$

The solution satisfying all the above constraints is called a feasible solution. Let C_k denote the completion time of the job k, (the moment of finishing the execution of all operations of a given job),

$$C_k = S_k + \sum_{i=1}^{r_k} p_k^i.$$

The length $C_{\max}(S)$ of a sequence S is called the time of executing all jobs:

$$C_{\max}(S) = \max_{k\in J}\ C_k.$$

The problem here consists of finding a feasible solution S^* with a smallest length sequencing $C_{\max}(S^*)$, also called the makespan.

2.2 Solution coding

Even though a solution to a given problem is S vector of the times of the beginning of the job's execution, this vector will not be a decisive variable in the presented algorithms. All the analyzed solutions will be generated by the midpoint algorithm which is later called a packing procedure, whereas a decisive variable will be a parameter steering the procedure later called a loading permutation.

Let $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ denotes a permutation of all n jobs of J set (loading permutation). The set of all possible loading permutations will be marked with Π. The packing procedure consists of n identical steps. In each of them, on the basis of a previously acquired partial solution, there is a sequencing of a following job. The sequencing of one job $k \in J$ means setting S_k as the beginning moment. In the ith step, the job $\pi(i)$ is sequenced in a way that the created partial (or the finality in the last step) sequencing was a feasible solution. During this job, the jobs' beginning moments, defined in previous steps, cannot be altered.

The effective application of the loading permutation requires applying a very specific method of coding a current schedule. Shown here is a sequence of m lists. The single element of each list is a pair (the beginning and the end) of moments defining a time interval of a non-stop usage of a machine. The l list includes a chronologically ordered set of all time intervals of a l machine usage. It must be noted that the set moment of beginning the execution of a sequenced job equals zero or some operation of a sequenced job is began exactly at the moment of finishing some other operation on the same machine (the job is moved towards the left side on the time axis). Thus, we can deduct from the above, that in order to find the smallest S_k moment of the sequenced k job, we must check the zero moment and all the moments resulting from initiating the $o_k^l \in O_k$ operation in all moments of slowing the m_k^l machine. We must pay attention to the fact that the beginning of k job in the latest of all analyzed moments always creates (the partial or the final) feasible solution. Thus, there is no danger of lack of possibilities to generate a feasible solution. It can be deducted from this statement that for each possible loading permutation there is exactly one feasible solution.

Let N denotes the biggest amount of operations in a given instance of a job, $N = \max_{k \in J} r_k$; $o \leq n \cdot N$. It must be noted that the number of possible initiating moments of k job should equal to not more than $o \cdot N$. Additionally, in the case where each operation of an exact k job is executed on a different machine the number will not exceed the o value. This situation can be observed in all the tested instances, thus, the analysis of calculating complexity will be extended by this specific case:

$$m_k^a \neq m_k^b, \quad 1 \leq a < b \leq r_k, \quad k \in J, \tag{1}$$

called later a special case. Moreover, it can be seen that in the described situation the value of N parameter is always not bigger than m; $N \leq m$.

The effective test of the sequencing acceptability assumes that the analysis of all checked moments must be carried out in a chronological order. For the analyzed S_k moment of a given o_k^l operation, we must exclude all the time intervals in a m_k^l list which finish before S_k^l moment of beginning this operation. In order to verify the proper (in the meaning of imposed synchronic constraints) location of o_k^l activity, it is enough to check whether the first from the left intervals begins not earlier than $S_k^l + p_k^l$ moment of finishing the o_k^loperation. During the test of all analyzed moments, the intervals in using a machine (for a given operation) are tested one after another and their number is not bigger than o. Thus, in the worst case the test of all moments of beginning the job requires not more than $o \cdot N$ analysis of intervals (of both excluding and introducing the constraints). Additionally, in a special case refer to the formula in Eq. 1 where this number is not bigger than o.

The calculating complexity of one step in a loading procedure in a general case equals the $O(oN) = O(nN^2)$. This is why the whole procedure has the complexity $O(n^2N^2)$. In a special case, the calculating complexity of one step equals $O(o) = O(nm)$ and the whole procedure has the complexity $O(n^2m)$.

The presented approach based on a loading function can be characterized by the following properties:

1. For each loading permutation, there is only one feasible solution generated.
2. The generated solutions can be characterized by a very high quality in the meaning of the value of goal function.
3. The number of different loading permutations is much smaller than the number of all active solutions.
4. Different loading permutations can generate the same solutions.
5. The possibilities of leaving the feasible solution even in case of the analysis of all possible loading permutations.

A more thorough analysis, examples and numerical tests proving the above-mentioned conclusions are presented in the work of Bożejko and Makuchowski [1]. They propose an additional application of not only the loading procedure but also its symmetrical equivalent which improves the effectiveness of an algorithm. It is caused by the fact that some of the instances, those difficult to be solved, become much easier instances when we look at their symmetrical, mirror equivalents.

Because of the fact that a decisive variable is a loading permutation, steering the nontrivial loading procedure, it is difficult to select some of its features and properties which will decide the quality of the final solution (e.g., the block properties). Thus, this is an ideal problem example of applying the genetic algorithm because these type of algorithms do not use the specific properties of the problem at all (or very rarely) and are based mostly on the value of the goal function of the tested solutions.

2.3 Test examples

All the numerical experiments presented in this work were carried out on the 40 literature test examples `la01-la40` which can be taken from the OR-Library [13]. These examples are dedicated to the classic job-shop problem and are considered by the scientists to be extremely difficult. Obviously, there are many other difficult sets of tests examples, nevertheless, for the purpose of this work the set proposed by Lawrence [8] seems to be large and differentiated enough. It is divided into eight groups with five instances each. In each group, all the examples possess the same size, more exactly, they have the same number of n jobs, m machines, and o operations. Moreover, the distinctive feature of these examples is the fact that the number of all operations equals exactly $o = n \cdot m$ and each job consists exactly of m operations $r_k = m, k \in J$ executed on different machines fulfilling the condition (1). In the latter part of the work, each group is defined by giving the size of an instance included in $n \times m$. The results presented in the tables which are related to the whole groups are average values of the corresponding values acquired in all the examples from the given group.

3 Genetic algorithm

The general idea of genetic algorithm's run is described in the work of Holland [7], and it imitates Darwin's theory of evolution encountered in the world of biology. Such algorithm simulates not only the environment but also the lives of virtual individuals. Each individual is identified with one solution and the fitness' quality is evaluated on the basis of the value of goal function. After terminating the simulation of a given number of generations, the algorithm finishes its run, and for the final solution the solution corresponding to the best individual which appeared in the whole simulation is taken. Properly fitted parameters of genetic algorithm, more specifically, properly fitted mechanism of inheriting with a selection promoting individuals with a demanded feature guarantee that in a simulated world evolution appears. Evolution is understood here as a tendency to generate generations with better fitted individuals. By altering the definition of fitness, thus, altering the method of evaluating a given individual, we can make a choice of evolution's direction. In practice, the fitting value of a given individual is a value of optimized goal function for a corresponding solution. In a life of a single generation, the least fit die mostly without progeny, whereas those best fit become parents for the next generation. The new created individuals inherit the genes (some attributes of solutions) of their parents. Thanks to this mechanism, the created solutions are combinations of the best solutions from the previous generation. In order to avoid the generation's degeneration in which all the generated individuals are similar to one another, a small mutation is applied. Owing to this mutation, the algorithm examines a new area of the solution space which promotes leaving the local minima and facilitates in generating the individual with some features lost in the generation.

In each genetic algorithm, we can distinguish the following base elements (see also Fig. 1:

1. *Generating starting off population:* generating the first generation's individuals,
2. *Selection:* selecting parents from the whole generation,
3. *Crossover:* generating of new individuals on the basis of their parents' genetic code,
4. *Mutation:* introducing small alterations in a genetic material of new individuals,
5. *Stop criterion:* defines a condition stopping the algorithm's work (more often it is a maximum number of simulated generations or the time of algorithm's run).

Defining the above-mentioned elements of GA algorithm is a basic problem with which a creator of a given implementation of an evolutional algorithm is confronted. Furthermore, we propose a universal method of choosing the most beneficial variants of separate elements.

3.1 Level of mutation

Before we move on to discuss separate methods of genetic algorithm auto-tuning, we will introduce the notion called the level of mutation. This notion is linked

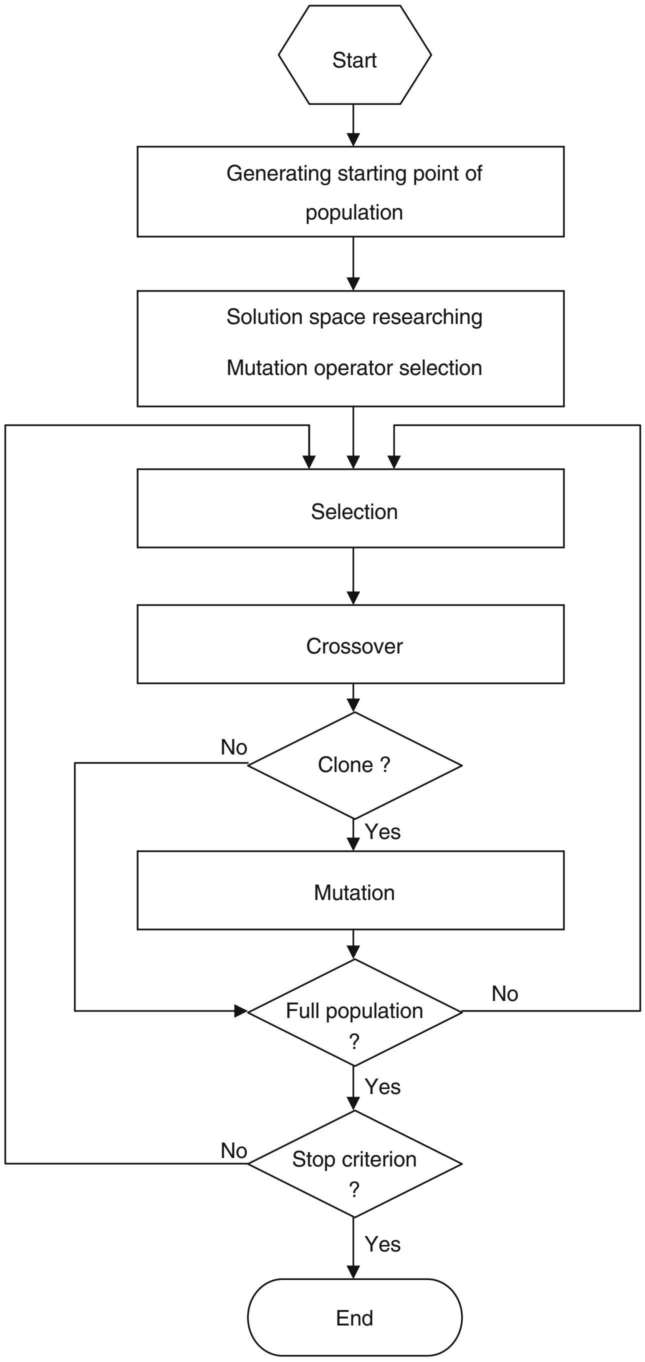
Start
Generating starting point of population
Solution space researching
Mutation operator selection
Selection
Crossover
Clone ?
No
Yes
Mutation
Full population ?
No
Yes
Stop criterion ?
No
Yes
End

Fig. 1 The outline of the proposed genetic algorithm

directly to the notion called possibility of mutation widely known in literature. The differences between them can appear subtle, nevertheless, they are crucial with respect to the research carried out in the latter part of the work. The L level of mutation denotes by definition a relative number of mutations in the whole generation, (*the number of all mutations* $= L \times$ *the number of all genes*), whereas the possibility of P mutation is by definition the possibility that a single gene will be mutated.

It must be observed that depending on the method of solution coding and applied mutation's operator, the mutated gene can alter its value in a limited way whereas in a random solution the value of this gene can have the values taken from a far more numerous set. In this case, even for $P = 1$ (each gene becomes mutated) there is a huge similarity between the original genotype and the mutated one.

The example of the above-mentioned phenomenon is the situation in which the whole genotype consists of one chromosome which is a permutation. If in such case the mutation means swapping of the two close-lying elements of permutation, then even during the mutation of each gene there is a great similarity between the x original and y-mutated permutation. The proof for that is the distance between those permutations $d(x, y)$ understood as the smallest number of neighboring alterations required to transform x permutation into y permutation. By definition, the distances and the number of executed mutations of even number of n elements in permutation, the maximum distance between x and y can reach at most n; $d(x, y) \leq n$, whereas the average distance between the z random permutation and x permutation is $\mathrm{AVE}(d(x, z)) = n \cdot (n-1)/4$ whereby the biggest possible distance between permutations is twice as big. This means that x permutation achieved by the mutation of each y permutation gene is similar to each other (where d is distance).

It can be deduced directly from the above property that mutation of previously mutated genes causes even more disturbances in an individual's genotype. This fact is not taken into consideration by P parameter which discusses the possibility of at least a single gene mutation. On the contrary, the level of mutation at a value bigger than 1 means that statistically in one individual there are more mutations than the number of genes it possess. However, in practice the optimal level of L mutation starts at a few percent and more or less equals P, and the deliberations for L value coming to 1 and exceeding 1 can be treated as only a theoretical forecast.

With an assumption that every mutation alters exactly k genes, we can define the relation of the possibility of P mutation to g number of all genes in a generation and L level mutation,

$$P(\eta, L) = 1 - \left(\frac{g-k}{g}\right)^{g \cdot L}. \tag{2}$$

Its results is presented in the 6th formula where the set L level mutation of the amount of genes in a generation does not influence greatly the P value. Moreover, for a small level of mutation (such levels are applied in a properly steered algorithms), its value equals almost exactly as the possibility of a mutation $P \approx L,\ L < 0, 1$. The additional subjective benefit of dependencies tested in a further part of the algorithm's parameters from the level of mutation instead of the possibility, is bigger clarity of the presented characteristics.

3.2 Choice of mutation's operator

In the beginning, we would like to remind that genetic algorithms do not work directly on solutions but on their coded representations (codes). Hence, there is a potential possibility in which individuals differentiate in codes but they represent exactly the same solution. In biology nomenclature, that would mean that there are different genotypes of exactly the same phenotype. In this case, there might appear a mutation in which a mutated individual, despite the code's alteration, generates exactly the same solution as the original individual. We will call such a mutation a barren one. Moreover, let ξ denotes the operator's effectiveness by a definition equalling the possibility that a randomly processed mutation will not be an idle one.

Despite the fact that mutation appears occasionally in genetic algorithm, it has an enormous meaning for the effectiveness of the whole algorithm. The basic aims of mutation are as follows:

1. Creating individuals partly different from their parents but in a crucial way, which allows moving the research into the new solution space.
2. Creating an individual with a negative trait not appearing in a generation which allows generating a solution that would be locally optimal even in case of losing some part of the solution in goal function generation.
3. Countermeasure to computability stagnation, in a situation when the whole generation is dominated by one kind of individuals which differ little from one another.

In order to become a parent (the condition necessary to reach aim 1), the value of an individual's fitness cannot be worse than the solutions in the generation. Hence, the mutation should generate individuals which will be different from the original one (in the meaning of the code), at the same time the fitness value of the mutated individuals should be similar to the fitness value of the individuals being mutated.

Differentiating the fitness function of the mutated individuals can be measured by the so-called autocorrelation of the solution space (Weinberger [15]). It is defined for the whole Π space, goal function $C(\pi)$, $\pi \in \Pi$, and distance measure $d(x, y)$, $x, y \in \Pi$,

$$\varrho(d) = 1 - \frac{\mathrm{AVE}\Big(\big(C(x) - C(y)\big)^2\Big)_{d(x,y)=d}}{\mathrm{AVE}\Big(\big(C(x) - C(y)\big)^2\Big)}, \tag{3}$$

where $\mathrm{AVE}((C(x) - C(y))^2)_{d(x,y)=d}$ denote the average value $(C(x) - C(y))^2$ of all pairs $x, y \in \Pi$ in which $d(x, y)$ it equals exactly d, whereas $\mathrm{AVE}((C(x) - C(y))^2)$ denotes the value analogous to all possible pairs. Value $\varrho(d)$ alters from 0 to 1 and defines the correlation of solutions with a d distance from each other.

From our point of view, a pair of individuals $x, y \in \Pi$, $x \neq y$, where y was created from the mutation of x individual, is distanced from each other by 1, in measure corresponding to a given mutation (this measure can be defined as a minimum number of mutation transforming an x individual into a y individual). It results in a statement that interests us, where the fitness function differentiation of the mutated individuals is closely linked to $\varrho(1)$ value of the proper distance measure. The $\varrho(1)$ value close to 1 means little differentiation, in the meaning of goal function's value, of the mutated individuals from the original individuals. On the contrary, $\varrho(1)$ value close to 0 means that there is no relation of goal function's value between mutated solutions and the original solution.

In order to define the approximate $\varrho(1)$ value denoted later by ϱ, we propose the set of $X = \{x_1, x_2, \ldots\}$ of the randomly chosen organisms and the set of $Y = \{y_1, y_2, \ldots\}$ of mutants where y_i was created by mutation of x_i organism. Thus, the ϱ value can be defined by:

$$\varrho(1) \approx \varrho = 1 - \frac{\mathrm{AVE}\Big(\big(C(x_i) - C(y_i)\big)^2\Big)}{2\Big(\mathrm{AVE}\big(C(x_i)^2\big) - \mathrm{AVE}\big(C(x_i)\big)^2\Big)}. \tag{4}$$

It must be noted that little effectiveness decreases greatly the roughness of the space solution (increases the ϱ quotient). As one of the criteria of the mutation operator's choice, we propose a modified roughness of the given operator's solution space defined by ϱ^* which is counted similarly to ϱ value with a difference that we do not take into consideration the solution in which the pairs of x_i and y_i organisms represent the same solution. Operators with high ϱ^* value stand a great chance that the organisms created by them (representing a completely new solutions) will qualify to a parent group. This means that the operators will efficiently carry out the given jobs.

To summarize, from the set of mutation's operators among operators with the biggest value of ξ parameter it is advised to choose the parameter with the biggest ϱ^* value. An additional suggestion is to make a single choice of an operator before launching the steered algorithm. In the opposite example, in the case of an on-line operator's choice, we must remember to alter the value of a mutation's possibility. It is needed because the optimal value for the possibility of mutation varies for different operators. Applying the automatic choice of the level of mutation described in the further work may facilitate this process.

4 Computational experiments

All the presented tests were carried out on a personal computer equipped with an Athlon 2000+ (1,667 MHz) processor in the multitask environment of Windows XP. All the described algorithms were programmed in a C++ language and compiled by Dev C++ in 4.9.9.1 version.

Because the decisive variable is a permutation (here, a loading permutation) to the test, we chose four commonly applied (for permutations) mutation's operators:

1. *Small swap* (SSw): the swap of two neighboring elements of permutation.
2. *Swap* (Swp): the swap of two different random elements of permutation.
3. *Insert* (Ins): one random element is taken out from the permutation and then put back in a random position (excluding the original position).
4. *Invert* (Inv): the sequence of elements in a random part of permutation is inverted.

The first step was to check the ϱ roughness, modified ϱ^* roughness, and ξ efficiency of the mentioned mutation's operators. The average values of the parameters for the separate groups of examples are presented in Table 1. After a thorough analysis, there are following conclusions to be drawn.

In conjunction with the increase of n number of jobs, the ϱ value has a decreasing tendency (for all the checked operators). Exactly the contrary influence on that parameter has m number of machines in a problem, together with its increase the ϱ value increases as well. However, the kind of applied mutation's operator has the decisive influence on the ϱ parameter. The most distinctive operator is SSw for which the generated space of solutions is far more smooth (big ϱ) than in the case of other operators. It is caused by two factors: (1) the SSw operator often alters only genotype of an organism not altering its phenotype (which is indicated by small ξ value) and (2) the SSw operator is a special case (the possibly smallest version) from each of the three operators. The sequence of all operators towards the roughness of the generated solution space (according to non-growing ϱ) is consequently: SSw, Ins, Swp, and Inv.

The analysis of the modified ϱ^* roughness shows a huge positive correlation to the ϱ value. All the properties observed for q refer also to the modified ϱ^* value. However, this time the influence of the mutation's operators has little meaning and ϱ^* value of the SSw operator does not diverge that far from the analogous value of the other operators. This phenomenon can be explained directly by the fact that a modified value of roughness of the space does not depend (contrary to non-modified value) on the effectiveness of the ξ operator.

While analyzing the effectiveness' values of particular operators, it is noticeable that ξ value depends strongly from the chosen operator. The SSw operator is of very low efficiency, and on an average of 35% of mutations, it does not alter the solution (despite the changes in loading permutation, its corresponding solution was not altered).

The second test shows the influence of mutation level value of the given operator on the quality of the solutions generated. The quality is understood here as

Table 1 The average value of ϱ, ϱ^* and ξ parameters for the tested mutation's operators

Group	ϱ				ϱ^*				ξ			
$n \times m$	SSw	Swp	Ins	Inv	SSw	Swp	Ins	Inv	SSw	Swp	Ins	Inv
10 × 5	0.56	0.37	0.43	0.31	0.42	0.29	0.36	0.24	0.72	0.92	0.90	0.93
15 × 5	0.66	0.41	0.46	0.34	0.48	0.32	0.39	0.28	0.64	0.93	0.89	0.94
20 × 5	0.70	0.44	0.51	0.38	0.54	0.42	0.44	0.33	0.61	0.94	0.88	0.94
10 × 10	0.55	0.30	0.37	0.29	0.36	0.24	0.27	0.22	0.70	0.92	0.88	0.93
15 × 10	0.62	0.35	0.39	0.29	0.43	0.28	0.32	0.24	0.66	0.94	0.89	0.94
20 × 10	0.64	0.35	0.41	0.34	0.44	0.33	0.35	0.27	0.61	0.94	0.89	0.95
30 × 10	0.68	0.39	0.45	0.32	0.45	0.35	0.36	0.27	0.58	0.96	0.89	0.96
15 × 15	0.60	0.30	0.34	0.30	0.39	0.27	0.30	0.25	0.69	0.94	0.90	0.95
Average	0.63	0.37	0.42	0.32	0.44	0.31	0.35	0.26	0.65	0.94	0.89	0.94

a relative error of an x solution towards the referential x' solution,

$$\rho = \frac{C_{\max}(x) - C_{\max}(x')}{C_{\max}(x')} \cdot 100\%. \qquad (5)$$

As a referential solution, we used the GASA algorithm solution described by Schuster and Framinan [12]. The tested algorithm was activated with the following parameters: the number of simulated generations, 1,000; the number of organisms, 50; the crossing operator, PMX; and the selection process used, the roulette method. The table presented in Fig. 2 shows an average ρ value of the quality of the generated solutions for the different L values of the mutation levels.

4.0.1 Evaluation of results

From the table presented in Fig. 2, the results show that by sequencing the operators of mutations according to the average ρ value (with an optimal level of mutation for a given operator), we were able to obtain the following sequence: SSw, Ins, Swp, and Inv. It must be noted that the sequence was already designated on the basis of ϱ^* parameter.

The second conclusion that can be drawn from the results is the fact that for the examples in which $n = 10$ (the examples are characterized by a relatively small solution space), the smallest ρ value that can be reached is slightly smaller than in the ρ value which corresponds to the biggest value of the L level. It is caused by the very narrow space of solutions in which random algorithms (checking relatively big set amount of random solutions) manage quite well and by the fact that the genetic algorithm with a big level of mutations becomes a random algorithm. Hence, it can be said that in an analyzed problem, using the approach based on a heavily compressed algorithm which reduces the solution space from $(n!)^m$ to $n!$ caused the result in which only in groups 20×5, 20×10, 15×15, 30×10 including bigger instances to show well all the properties of the HGA algorithm.

The third observation is the fact that the kind of an applied mutation's operator has no crucial influence on the efficiency of an algorithm. From the table in Fig. 2, we can draw the conclusion that for the optimal levels of mutation (different for every operator) the quality of generated solutions is approximate. At the same time, it can be noted that badly adjusted level of mutations lowers the level of generated solutions in a significant way. Therefore, a special way of automatic adjustment is presented in the further work.

From the results presented here, especially from the evaluation of mutations' operators which are almost independent from the instance and from the relatively low influence of the mutation's operator on the efficiency of algorithm, the conclusion can be drawn that there is no need of automatic adjustment of the operator with on-line method (during the run of an algorithm). In a final conclusion, for a specific genetic algorithm dedicated to a given problem, we propose carrying out only one initial off-line test which will result in choosing one, based on a ξ efficiency and modified ϱ^* roughness, potentially best mutation's operator to be invariably applied during the whole run of an algorithm.

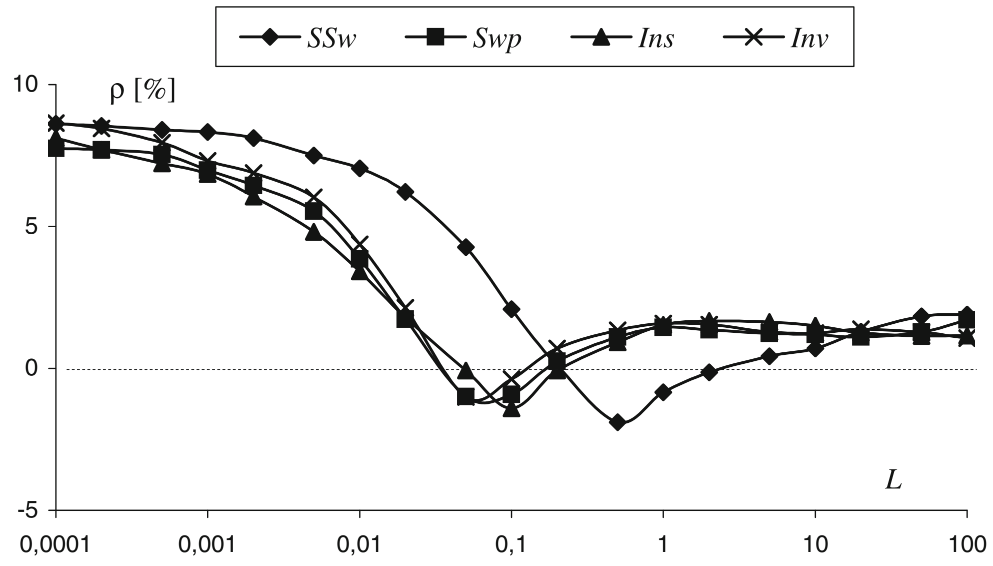

Fig. 2 The correlation between the ϱ average value and the level of L mutation for the tested mutation's operators

4.1 Choice of mutation's probability

From the research carried out in a previous point, we can see that some of the conclusions refer directly to the level of mutation. The most important is the fact that the level of mutation has a decisive influence on the efficiency of the algorithm. Besides, the optimal value of the mutation's level depends not only on the applied operator but also on the problem's data (view Fig. 3). The relations of the optimal value of mutation's level are easy to be forecasted, that is: together with an increase of the space roughness of the given operator (q decreasing) the value of the optimal mutation's level increases as well.

In order to make all the phenomena from taking place in a genetic algorithm, more understandably in Fig. 4, we show the run of an algorithm with too low, relatively good, and too big level of mutation.

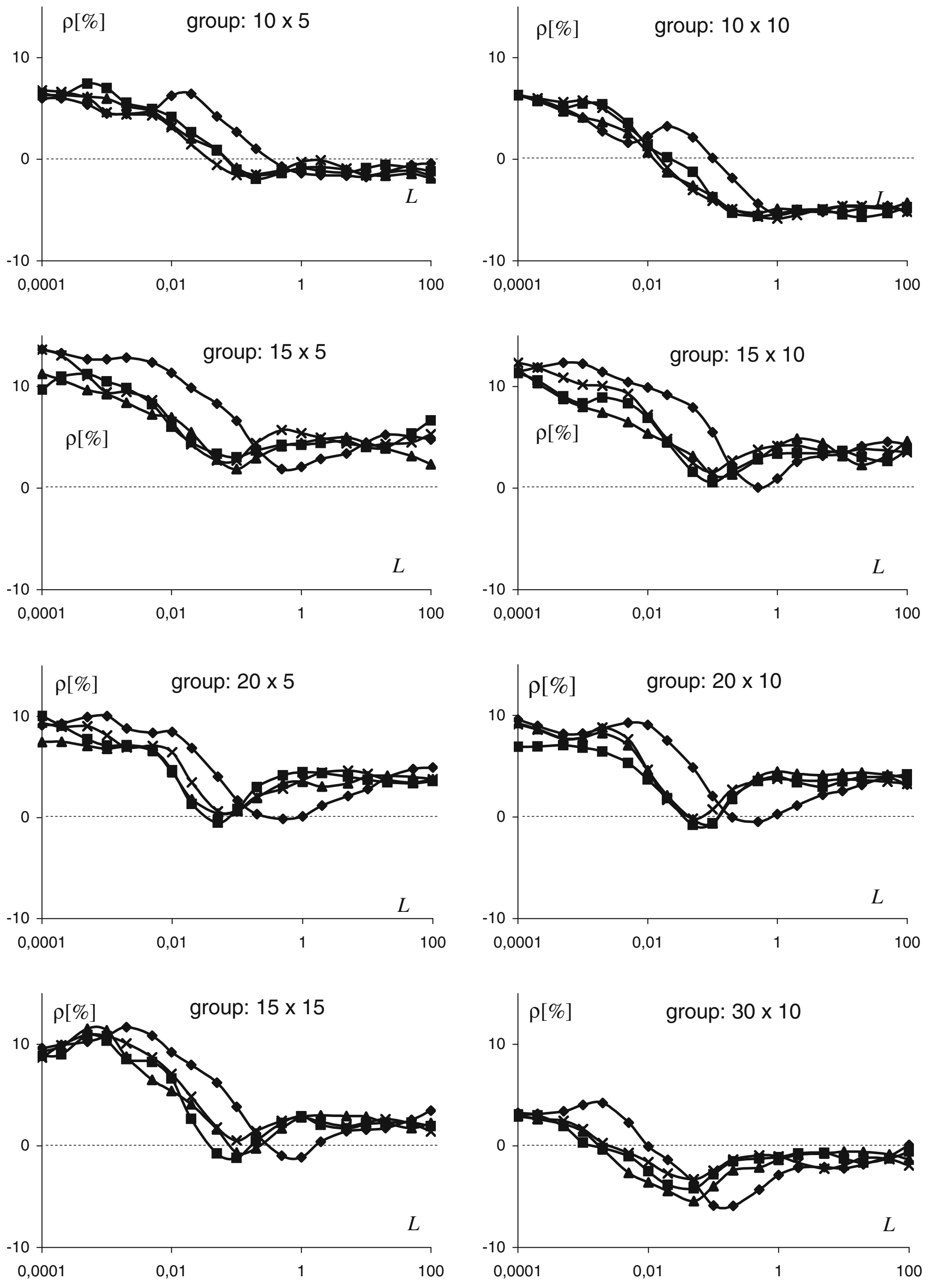

Fig. 3 The correlation between the ϱ average value and the level of L mutation

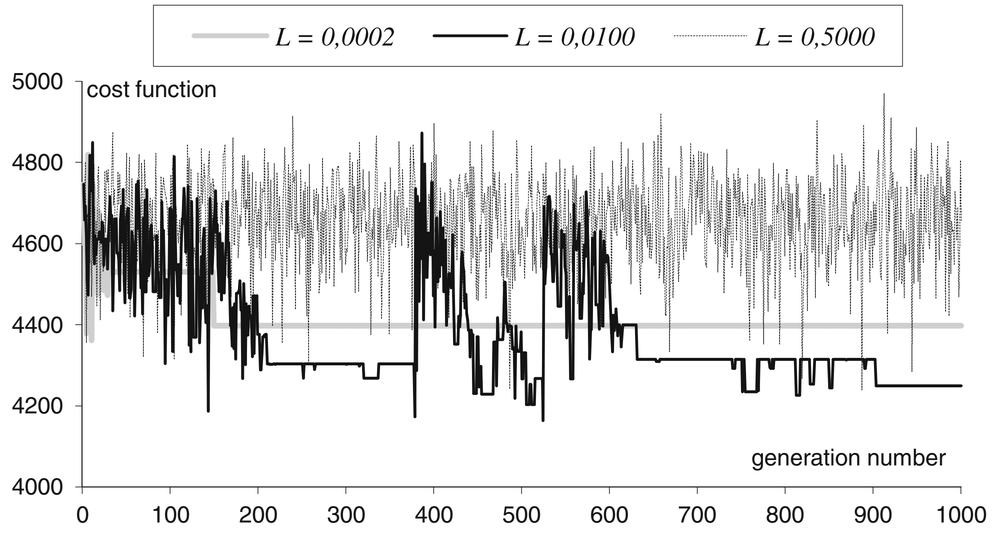

Fig. 4 The run of an exemplary genetic algorithm for the different levels of mutation. The *LA31* example

In the presence of too low level of mutation in HGA algorithm, there appears a counting stagnation (view the first run in Fig. 4). It is caused by the fact that the generation is dominated by so-called super organisms not only in number but in the quality as well. These organisms are practically identical, differentiated in a way that does not influence the overall result and the goal's function criterion. The super organisms are the best in the meaning of fitness function's value that very often it comes to their crossing which results in creating a new super organism in the net generation. All the other crossings in a generation generate offsprings weaker in the meaning of fitness function's value. In the consequence, after the process of natural selection the new generation is again dominated by the new super organisms identical with the previous generation. In this situation, there are only three possible scenarios of the simulated life: (1) some mutated organism will be better fit than the super organisms (with regard to the fact that super organisms are characterized by high fitness function and low mutation appearance in a generation, this alternative is less probable), (2) in a new generation there are no super organisms (this variant is of little probability as there is a big possibility of crossover of the two super organisms), and (3) the new created generation will be exactly the same dominated by super organisms as the previous generation (the most probable variant). Thus, there appears counting stagnation, the algorithm reached some kind of local optimum with a little chance of its leaving.

The case where the level of mutation is proper, that is, big enough not to allow the counting stagnation to appear in the process and low enough to allow the offspring inherit the traits of their parents, is presented in Fig. 4. The graph shows basic phenomena appearing in a properly steered genetic algorithm. These are: the fast descending of an algorithm to a local minimum and effective diversification of calculations shown in a skipping found local minima.

Because of the impossibility to find an optimal value of the level of mutation in an analytical way with the knowledge we currently have, the only method to determine it lies in numerical experiments. With regard to the fact that for different instances the optimal mutation level value is different, the choice of the proper level must be made for each instance separately or change it during on-line work while observing the run of the steered algorithm. Nevertheless, in this work the proposed method of mutation is slightly different from its classic original. To be more precise, instead of causing the genes' mutation with a fixed (permanently or dynamically changing) probability, an initial analysis of the population is proposed and on its basis we determine the organisms which will be mutated. This method not only determines the amount of mutations in a generation, but also appoints the organisms which should undergo the process of mutation.

The proposed strategy is based on two simple observations:

1. In case when the whole generation is strongly diversified, there is no need of an individual's mutation. In this example, mutation deprives individuals of the inherited traits.
2. In case of a few super organisms appearing in a generation, the mutation must be carried out. This kind of mutation secures avoiding the stagnation of calculations through eliminating of identical organisms. Moreover, it allows fast reaching the local optimum (in case when super organisms represent

the solution close to local optimum). Additionally, it allows easy leaving of the local optimum.

The strategy of proceedings, thus, seems to be clear. First, all the clones must be identified, secondly, they must undergo the mutation process. We define a clone as an organism similar in a way to the other organism existing in a given generation. This approach, although based on intuition has no equivalent in a natural world. In implementing the strategy proposed above, the problem to be determined is, among others, the method of detecting clones.

In a problem considered here, the application of solution coding in the shape of loading permutation impedes the effective comparison of the solutions (the comparison based on the loading permutation without the compressing procedure inducement). Therefore, in order to detect clones the simplest method was applied, that is through comparison of the organisms' fitness function's value. Organisms with the same value of fitness function are treated as clones. Moreover, clones selected for mutation should be mutated effectively in a way that the mutation alters not only the genotype but also the phenotype of an organism. In order to do that, we can for instance control the value of the fitness function of the mutated organisms.

The side effects resulting from clone detection and from the testing of the alterations appearing in an organisms being mutated on the basis of the goal function value of the corresponding solutions, is the possibility to state whether two organisms are identical, despite different phenotypes and additional time consuming examinations calculating goal function value of the muted clone. The first from the above mentioned phenomena occurs relatively rarely and does not cause any serious disorders in an algorithm, whereas the second phenomenon definitely slows down the algorithm's run. The strategy proposed earlier on the organism's mutation (its mutation until reaching the alteration in goal function value), termed here as a $FULL$ strategy, is very slow. The acceleration of the algorithm's run can be reached by reducing it to a strategy ($SINGLE$) with a single mutation and a single counting of the modified value of the fitness function (disregarding whether its value has changed or not). The last strategy which does not practically slows down the work of an algorithm is called a $BLIND$ strategy, which resulted in a single mutation of a given organism without recounting the fitness function value. The organism mutated in this way undergoes the selection process according to the original value of the fitness function.

4.1.1 Numerical examinations

The first step of the test serves a purpose of generating the run of genetic algorithm, which is the drawing of the value of the solution goal's function corresponding to the best organism in a current generation. Figure 5 shows an exemplary run of a genetic algorithm with an automated mutation and the SINGLE strategy. Additionally, for a comparison it shows the run of an algorithm with a classic mutation with a properly adjusted level of mutation. From the two runs, we can see that genetic algorithm with a automatic mutation generates solutions statistically better than in case of a fixed (properly adjusted) level of mutation. The domination of the automated choice of mutation over its classic equivalent is visible also when an algorithm reaches local minimum. In the case of a classic mutation, the algorithm is dominated through several iterations by

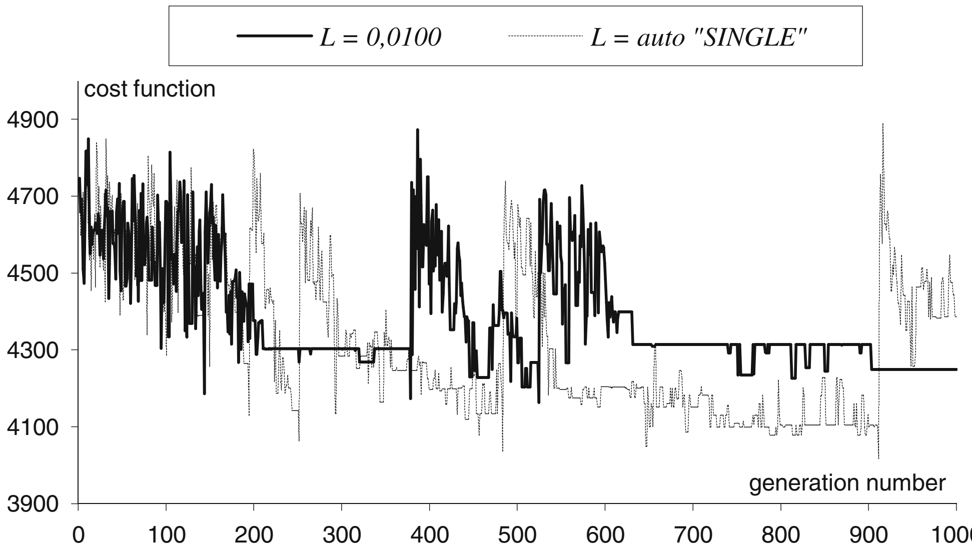

Fig. 5 The run of the exemplary genetic algorithm with an optimal and automatically adjusted value of the level of mutation. The $LA31$ example

Table 2 The allotted level of L^* mutation and corresponding with it average ρ (%) quality of generated solutions of the tested mutation' operators

Group $n \times m$	Optimal mutation level L^*				Average deviation ρ for L^*			
	SSw	Swp	Ins	Inv	SSw	Swp	Ins	Inv
10 × 5	2.00	0.20	0.10	0.10	−1.91	−2.63	−1.56	−1.47
15 × 5	0.50	0.10	0.10	0.10	0.31	1.94	1.29	1.77
20 × 5	1.00	0.05	0.05	0.05	−0.28	−2.01	−0.16	−1.22
10 × 10	1.00	0.50	0.20	1.00	−5.40	−5.29	−5.29	−5.47
15 × 10	0.50	0.10	0.10	0.10	−1.39	0.09	−0.73	0.78
20 × 10	0.50	0.05	0.10	0.05	−1.22	−2.69	−2.10	−2.15
30 × 10	0.20	0.05	0.05	0.05	−5.68	−4.69	−6.54	−3.88
15 × 15	1.00	0.05	0.10	0.10	−1.74	−1.86	−2.06	−0.56
All	0.50	0.05	0.10	0.05	−1.72	−0.83	−1.27	−0.86
Average	0.84	0.14	0.10	0.19	−2.16	−2.14	−2.14	−1.53

the same solutions (fragmentally even line of a table), this effect almost never occur by the automated mutation.

The second test serves a purpose of comparing the average value of ρ error generated by the genetic algorithm being steered by the adjusted L^* level of mutation and by the automatic mutation using the SINGLE, FULL, and BLIND strategies. The experimentally adjusted L^* level of mutation comes from the set $L^* \in \{0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, \text{and } 100\}$ as a level for which a single run of an algorithm appeared to be the most beneficial. With regard to the fact that the value of L^* level is denoted experimentally, it is only a rough approximation of the optimal L^{**} level value.

The discussed examinations were carried out for each of the tested mutation's operators, and all the acquired results were placed in Tables 2 and 3. As follows, Table 2 includes the value of the optimal L^* level and the average ρ quality for the algorithm of mutation at the L^* level whereas Table 3 shows the average ρ values of the algorithm with an automatic mutation of the FULL, SINGLE, and BLIND types. The ρ value is counted from the formula presented in Eq. 5 toward the solution acquired through literature (Schuster and Framinan [12]) and GASA algorithms. Moreover, the last *all* and *average* lines included in Table 2 need to be commented. The first one includes the values of the parameters obtained when the level of mutation was adjusted for all groups. The last line includes the average parameters obtained in a case when the level of mutation was adjusted individually for each group.

4.1.2 Results' evaluation

In the first part of the chapter concerning the choice of mutation's operator, we pointed out the main goals of applying mutation in genetic algorithms. Hence, it can be clearly seen from the observation of exemplary run of the algorithm with an automatically adjusted mutation (Fig. 5) that in the presented example it satisfies all the requirements concerning proper mutation. The main disadvantage of the proposed approach appeared to be the fact that there are operators of the mutation (in a presented case the SSw operator) which produces mere results of the presented automatic choice of mutation. However, for the other three Swp, Ins, and Inv operators, the results of proposed automatic mutation were very good.

By restricting to the analysis of the above-mentioned three operators of the mutation, from the comparison of Tables 2 and 3, it results that in a given problem all the variants of the automatic mutation is statistically

Table 3 The average ρ (%) quality of generated solutions for the tested methods of automatic mutation of the tested operators

Grupa $n \times m$	FULL mutation				SINGLE mutation				BLIND mutation			
	SSw	Swp	Ins	Inv	SSw	Swp	Ins	Inv	SSw	Swp	Ins	Inv
10 × 5	2.25	−1.79	−1.84	−1.68	1.61	−1.77	−2.47	−1.99	6.12	−2.04	−1.96	−1.65
15 × 5	6.45	1.16	1.29	2.87	5.64	0.69	1.46	2.72	6.14	0.64	1.06	0.32
20 × 5	2.61	−1.77	−1.80	−0.71	2.77	−0.68	−1.68	−1.82	5.13	−1.80	−1.67	−0.95
10 × 10	−0.25	−3.08	−2.39	−3.94	−0.90	−3.93	−3.53	−3.89	−0.77	−2.71	−3.44	−3.39
15 × 10	5.51	0.46	−0.74	−0.79	5.91	−0.34	2.16	0.69	5.63	0.62	1.04	0.33
20 × 10	1.90	−2.11	−1.99	−1.08	5.28	−0.56	−0.90	−1.58	3.94	−1.33	−0.93	−0.56
30 × 10	−2.96	−6.47	−6.19	−4.90	−1.03	−6.15	−7.55	−5.33	0.17	−6.79	−5.05	−5.80
15 × 15	4.82	−0.45	−2.35	−2.19	5.09	−1.93	−0.73	−0.15	7.05	−0.77	0.67	−0.97
All	2.54	−1.76	−2.00	−1.55	3.05	−1.83	−1.66	−1.42	4.18	−1.77	−1.29	−1.58

better than the best adjusted classic mutation. What is more, in case when the best possible level of mutation is adjusted individually to each of the groups of examples, the algorithms with automatic mutation perform only little less satisfactory results. It must be stressed that the last comparison shows the automatic mutation in a less positive position. It is an effect of the fact that presented test showed a single run of all algorithms and the acquired results are partially random. In the algorithm with a fixed level of mutation it happens that for some level of L^* mutation (statistically weaker than L' level closer to optimal L^{**} level) the ρ values from single runs will be better than for the L' level. In that case in order to carry out the comparative assessment the best results are taken into consideration, that is the runs for L^* level instead of the runs for L' level. This experiment eliminates single snags of classical algorithm for L' level and additionally chooses successful cases of runs with other mutation levels especially at L^* level. In order to assess the algorithm with an automatic mutation, we take only one run that could be worse or better than the average.

4.2 Choice of crossover operator

The task of the crossover operators is creating new individuals from the parents' solutions. Despite the fact that in the process of creating the new individual there is some randomization, the properly adjusted crossover operators guarantee that the newly created individuals inherit genes (some properties of the represented solutions) after their parents.

Some analogy explaining the meaning of crossover is comparing an individual to the idea generating solution and the whole population to the set of solutions. In this analogy, a new individual is nothing more than an innovative idea. By giving a careful consideration of what is an innovative idea, we come to conclusion that more often these are combinations of concepts previously used and proved in the past. Thus, the new generated individuals are some kind of a crossbreed of the best solutions from the past (classically a crossbreed of the best adapted individuals from the previous generation).

The method of the purpose of the crossover operators seems to be clear. The better operator, the better generating of individuals from the given original population in the meaning of fitness function value. Because classic operators link a pair of individuals together, in order to create an individual that will be a crossbreed of more than two solutions we must carry out a few iterations of crossovers. The second criterion of the operators' evaluation is the speed (counted in algorithm's generations) of the found local minimum. On the basis of the mentioned observations in order to evaluate the quality of the crossover operator we propose to take goal function value of the found solution in a test run of the genetic algorithm without mutation with a set of relatively small number of simulated generations. This evaluation promotes the operators which quickly find good local minimum. By increasing the number of simulated generations in a test run of algorithm, we decrease the weight attached to the speed of local minimum finding instead concentrating on its quality. It allows to arrive at the desired priorities between the speed and the value of the solutions we are looking for.

Because in a genetic algorithm, similarly like in an ecosystem, there are many subtle bonds between its elements, it seems that the most desirable method would be taking into consideration the choice of crossover operator from earlier adjusted mutation's parameters. The second—very intuitive method of crossover operator choice means allowing the run of a fully steered genetic algorithm for some number of iterations for all tested crossover operators at some amount of tested examples. This test is nevertheless time consuming, thus, we propose only a single run for a newly appointed method.

4.2.1 Numerical examinations

The following crossover operators will be the subject of numerical exercises: {*LX, RX, LRX,* and *MX* }. The first two are one-point operators, whereas the following two are two-point operators. Before a more detailed description of the separate operators, we would like to define the base operations on sequences: $\oplus$ operator linking two sequences in one through positioning one by one all elements of the sequence from the left side of operator, the $\ominus$ operator creating a sequence from elements of the sequence from the left side of the operator smaller by elements of a sequence from the right side of an operator, and both operators are left-associative and have the same priority of arranging the order of doing the jobs. Moreover, for any sequence $X = (x_0, x_1, \ldots, x_n)$ symbol $X_{[j,k)}$, $0 \le j \le k \le n+1$ denotes the sequence $X_{[j,k)} = (x_j, x_{j+1}, \ldots, x_{k-1})$. Denotations of parental loading permutations are as follows: $A = (a_0, a_1, \ldots, a_{n-1})$ and $B = (b_0, b_1, \ldots, b_{n-1})$. Additionally, let j denotes the point of crossover for the one-point operators and let k denote the second point of crossover for the two-point operators. Through applying the introduced notation, the description of the tested crossing operators can be presented in brief in an equation:

$$\mathrm{LX}(A, B, j) = A_{[0,j)} \oplus \left(B \ominus A_{[0,j)}\right), \tag{6}$$

Table 4 The evaluation of the crossover operators based on ρ (%) from the test run of algorithm without mutation and ρ (%) quality of the proper algorithm

Group	ρ (%) iter = 10				ρ (%) proper run			
$n \times m$	LX	RX	MX	LRX	LX	RX	MX	LRX
10 × 5	6.73	9.20	10.36	10.02	−2.62	−1.11	−0.55	−0.92
15 × 5	13.65	14.49	12.68	12.15	1.54	1.49	3.14	0.84
20 × 5	11.67	12.77	12.44	11.89	−1.85	−0.74	−1.75	−0.14
10 × 10	6.32	6.10	6.20	3.31	−3.82	−3.46	−2.92	−3.81
15 × 10	12.35	13.04	11.97	13.04	2.28	0.09	0.82	1.80
20 × 10	10.29	13.54	10.65	11.55	−1.05	−1.04	−1.14	0.05
30 × 10	5.92	6.24	7.14	7.62	−8.75	−6.06	−6.40	−7.01
15 × 15	12.73	11.38	10.13	13.45	−0.79	−1.98	0.55	−0.78
All	9.96	10.85	10.20	10.38	−1.88	−1.60	−1.03	−1.25

left side of the created permutation is copied from A parent and the other elements of permutation are arranged in the sequence of appearing in B parent.

$$\mathrm{RX}(A, B, j) = \left(B \ominus A_{[j,n)}\right) \oplus A_{[j,n)}, \tag{7}$$

right side of the created permutation is copied from A parent and the other elements of permutation are arranges in a sequence of appearing in B parent.

$$\mathrm{LRX}(A, B, j, k) = A_{[0,j)} \oplus \left(B \ominus A_{[0,j)} \ominus A_{[k,n)}\right) \oplus A_{[k,n)}, \tag{8}$$

Left and right side of the created permutation are copied from A parent whereas the middle part is completed with the missing elements in the sequence of appearing in B.

$$\mathrm{MX}(A, B, j, k) = \left(B \ominus A_{[j,k)}\right)_{[0,j)} \oplus A_{[j,k)} \oplus (B \ominus A_{[j,k)})_{[j,j-k+n)}, \tag{9}$$

middle part of the created permutation is copied from A parent whereas the left and the right side of permutation is completed with the missing elements in the sequence of appearing in B parent.

In Table 4, ρ quality values of solutions acquired for the entry and proper run are presented. The entry run is a trial of evaluating the use of separate crossover operators whereas in the proper run there appears an exact verification of the evaluation. The entry run is acquired by initiating the genetic algorithm without mutation and with a small number of ten generations whereas the verifying run is acquired through running wholly steered genetic algorithm with a INS type mutation being adjusted automatically according to the SINGLE strategy with a 1,000 number of generations. In Table 5, ρ values of acquired solutions with the work of a genetic algorithm with a mutation described above for the first ten and 100 generation are presented.

4.2.2 Evaluation of results

From the results presented in Table 4, on the basis of the entry run, we can expect that in the presented problem the most promising will be the LX crossover operator. This theory is confirmed later by the results of examinations in which a fully steered genetic algorithm with the above-mentioned LX operator generates the best solutions among the analogical algorithms with the other operators. However, the following suppositions based on the entry run with the reference to all other operator has not been confirmed in a verifying test. It is partially caused by the fact that all the operators having been tested are approximately similarly good in the sense of efficiency of the tested algorithm and finally only an accident decides which of them is better in a

Table 5 The evaluation of the crossover operators on the basis of ρ (%) value from the trial run of a fully steered algorithm

Grupa	ρ (%) iter = 10				ρ (%) iter = 100			
$n \times m$	LX	RX	MX	LRX	LX	RX	MX	LRX
10 × 5	7.09	6.37	5.96	6.47	−0.29	−0.38	2.60	0.14
15 × 5	13.42	12.93	10.35	11.10	6.33	7.11	5.53	2.90
20 × 5	11.05	9.97	10.12	9.47	3.00	4.94	4.12	2.87
10 × 10	5.81	6.65	3.05	6.48	−0.25	−2.12	−0.60	−1.90
15 × 10	13.13	12.49	12.52	10.84	5.53	6.13	4.49	7.10
20 × 10	11.53	10.49	11.05	10.84	5.36	3.93	3.22	5.23
30 × 10	5.30	4.67	5.15	6.69	−1.81	0.06	−1.44	−0.47
15 × 15	12.19	13.42	10.45	12.42	3.53	3.99	4.42	3.42
All	9.94	9.62	8.58	9.29	2.67	2.96	2.79	2.41

given circumstances. In order to confirm this statement, we carried out some additional verifying tests with the application of mutation with an even level. The acquired result (not presented in this work) showed other hierarchy of usefulness of individual operators which proves the theory about the incidental value of the tested operators.

The weak point of the test evaluating the tested crossover operators does not result from its superficiality but from the fact that the tested operators are similarly good, thus, in such a case the other elements of the genetic algorithm such as the number of individuals in a generation, the selection type, the number of simulated generations, or some randomness have a decisive influence on the relative value of the separate crossover operators. The notion of relative value of the crossover operators is understood here as a value of the generated solutions by the genetic algorithm with a checked crossover operator in reference to the analogical value of the other operator.

In a given problem, among the tested crossover operators, it is impossible to state with no doubt which of them should be applied. The crossover operator, which in some circumstances is better from the group of accessible operators, might appear to be less effective from the others with an alteration of the algorithm's steering parameters. The example of this case are the results of the proper algorithm presented in Table 4 and the result with a limited number of ten and 100 iterations presented in Table 5. For the number of iterations equalling to ten, statistically the best operator appeared to be MX, for the number of 100 generation the best operator appeared to be LRX whereas for the simulation with 1,000 generations the best results were acquired with the LX operator.

4.3 Scaling of the fitness function value

Natural selection existing in a real ecosystem is to certainly extend an effect of a random choice; nevertheless, it promotes individuals who are better fit than the others. The fitness in nature can be interpreted as a sum of elements which have influence on the survival of a given individual in the environment, e.g., immunity to diseases, ability to get food, ability to avoid predators, etc. It is clear that better-fit individuals stand a greater chance to survive and bear more offsprings. In genetic algorithms, the value of the goal function of the fitted individual is calculated on the basis of the goal function value of the corresponding solution.

The selecting procedure can be carried out in many different ways, however, the most popular one is a method using properly calibrated roulette wheel. In this method, every individual gets some sector of a wheel with a surface proportional to the value of the fitness function of this individual in a way that all individuals in a generation fill in all the gaps in the surface of the wheel. Then, the drawing of an individual is made by turning the wheel by the random angle from the bracket $[0, 2\pi)$ and choosing the individual corresponding to the chosen fragment of the wheel.

In selecting methods, some function f transforming directly the goal function values of solution into the fitting values of a given individual is often applied. More often, it is a line function represented as $f(x) = ax + b$. In the book of Goldberg [3], the need of an automated fitting of a and b parameters for each generation is explained in a following way: it happens often that in the beginning of the run of a population there appear a few above average individuals, whereas the others belong to the average category. If a normal method of selection was applied, the above average individuals would take a huge part in the finished population of one generation, which would be an undesirable phenomenon, a major cause of the premature convergence. In a further phase, the problem is quite different. It might happen that at the end of the run the population is quite diversified but the average indicator of adjustment is placed not far from the average. If we do react to this phenomenon, in the result, the average and the best individuals will have the same amount of progeny in net generations which will lead to the rule of the best-fit individuals surviving among the average into a randomly mistaken way.

As an antidote to the two abovementioned wrong situations, we propose an automated fitting of a and b parameters of f function. The proposed choice of parameters means that the f_{avg} average value of the fitting function equals the c_{avg} average value of the goal function, and the fitting value of the f_{best} individual equals C_{mult} of the average fitting value,

$$\begin{cases} f_{\mathrm{avg}} = a \cdot c_{\mathrm{avg}} + b \\ f_{\mathrm{best}} = a \cdot c_{\mathrm{best}} + b \\ f_{\mathrm{avg}} = c_{\mathrm{avg}} \\ f_{\mathrm{best}} = C_{\mathrm{mult}} \cdot f_{\mathrm{avg}} \end{cases} \tag{10}$$

It must be noticed that in the above dependencies on the basis of c_{avg} and the best c_{best} goal function and the taken C_{mult} parameter we can unequivocally designate a and b parameters. Unfortunately, in some situations this scaling could lead to designating minus values of the fitting function to the weakest individuals (the fitting value of a given individual is a non-minus figure by definition). In order to avoid the above mentioned situations in Goldberg [3], we propose an additional modification of C_{mult} parameter, in a way that the

Table 6 Results of the proposed HGA with automatic adjustment compared with GASA of Schuster and Framinan [12], and TS of Framinan and Schuster [2]

Inst. name	$C_{\max}^{\mathrm{Ref}}$	GASA(Schuster)			TS(Schuster)			HTS		
		$C_{\max}^{\mathrm{GASA}}$	t^{GASA}	ρ^{GASA}	$C_{\max}^{\mathrm{TS}}$	t^{TS}	ρ^{HGA}	$C_{\max}^{\mathrm{HGA}}$	t^{HGA}	ρ^{HGA}
la01	971	1,037	23	6.80	1,043	1	7.42	971	8	0.00
la02	937	990	24	5.66	990	0	5.66	961	8	2.56
la03	820	832	24	1.46	832	1	1.46	820	8	0.00
la04	887	889	25	0.23	889	0	0.23	887	7	0.00
la05	777	817	24	5.15	817	0	5.15	777	7	0.00
la06	1,248	1,339	80	7.29	1,299	2	4.09	1,248	18	0.00
la07	1,172	1,240	70	5.80	1,227	5	4.69	1,207	18	2.99
la08	1,244	1,296	72	4.18	1,305	2	4.90	1,289	18	3.62
la09	1,358	1,447	83	6.55	1,450	3	6.77	1,371	18	0.96
la10	1,287	1,338	70	3.96	1,338	4	3.96	1,327	18	3.11
la11	1,621	1,825	170	12.58	1,737	11	7.16	1,724	33	6.35
la12	1,434	1,631	164	13.74	1,550	10	8.09	1,542	32	7.53
la13	1,580	1,766	183	11.77	1,701	17	7.66	1,685	33	6.65
la14	1,610	1,805	176	12.11	1,771	10	10.00	1,677	33	4.16
la15	1,686	1,829	167	8.48	1,808	7	7.24	1,780	33	5.58
la16	1,575	1,637	39	3.94	1,637	1	3.94	1,575	26	0.00
la17	1,371	1,430	42	4.30	1,430	1	4.30	1,371	26	0.00
la18	1,417	1,555	42	9.74	1,555	1	9.74	1,485	25	4.80
la19	1,482	1,610	40	8.64	1,610	1	8.64	1,482	26	0.00
la20	1,526	1,693	45	10.94	1,705	1	11.73	1,526	26	0.00
la21	2,030	2,182	147	7.49	2,242	5	10.44	2,098	64	3.35
la22	1,852	1,965	135	6.10	2,008	3	8.42	1,912	63	3.24
la23	2,021	2,193	136	8.51	2,093	5	3.56	2,093	65	3.56
la24	1,972	2,150	133	9.03	2,061	5	4.51	2,054	64	4.16
la25	1,906	2,034	142	6.72	2,072	6	8.71	1,976	62	3.67
la26	2,506	2,945	332	17.52	2,664	14	6.30	2,649	120	5.71
la27	2,675	3,036	311	13.50	2,968	27	10.95	2,880	123	7.66
la28	2,552	2,902	324	13.71	2,886	24	13.09	2,756	121	7.99
la29	2,300	2,617	311	13.78	2,671	12	16.13	2,592	119	12.70
la30	2,452	2,892	346	17.94	2,939	12	19.86	2,743	117	11.87
la31	3,498	4,298	957	22.87	3,822	151	9.26	3,743	282	7.00
la32	3,882	4,686	869	20.71	4,186	176	7.83	4,275	288	10.12
la33	3,454	4,214	860	22.00	3,869	120	12.02	3,835	284	11.03
la34	3,659	4,401	968	20.28	3,957	102	8.14	3,909	287	6.83
la35	3,552	4,299	897	21.03	3,908	120	10.02	3,990	287	12.33
la36	2,685	2,949	203	9.83	2,993	9	11.47	2,858	136	6.44
la37	2,831	3,216	192	13.60	3,171	7	12.01	3,093	137	9.25
la38	2,525	2,762	202	9.39	2,734	6	8.28	2,695	132	6.73
la39	2,687	2,908	195	8.22	2,804	9	4.35	2,697	136	0.37
la40	2,580	2,950	214	14.34	2,977	12	15.39	2,594	138	0.54
Average			235.93	10.50		22.58	8.09		86.15	4.57

t in second, ρ in percents

smallest value of the fitting function corresponding with the worst solution in a generation was not less than 0.

In the algorithms presented, we propose a little bit different strategy to avoid minus values of the fitting function. Once designated in a generation, a and b values according to the formula (10) do not undergo modifications, and any possible minus values of the fitting function are transformed into the values equalling 0. Finally, we thus propose to replace a line f function with its f' equivalent

$$f'(x) = \max\{0, ax + b\} \tag{11}$$

where a and b values are designated depending on Eq. 10. Selection method can be regulated by the C_{mult}

parameter alteration. For the $C_{\text{mult}} = 1$, all individuals stand the same chance to be chosen. By gradual increase of the parameter's value, better individuals are promoted (better in the sense of goal function value of the corresponding solution).

4.4 Numerical verification

Here, we will provide a final test of all the described automatic steering application in the HGA. Obtained results will be compared with GASA as proposed by Schuster and Framinan [12]. The following parameters of the proposed HGA have been used: 20 individuals in a generation, 10,000 generations, LX crossover operator, SWP mutation with the BLIND mutation strategy, and $C_{\text{mult}} = 5$.

Numerical tests of the proposed HGA algorithm consist of generated solutions quality and working time comparison with other algorithms (GASA [12], TS [11]) taken from the literature. For each instance, a relative deviation of a scheduled length to a reference solution taken from Bożejko and Makuchowski [1] was calculated:

$$\rho^{\text{Alg}} = \frac{C^{\text{Alg}}_{\max} - C^{\text{Ref}}_{\max}}{C^{\text{Ref}}_{\max}} \cdot 100\%,$$
$$\text{Alg} \in \{\text{GASA}, \text{TS}, \text{HGA}\}. \qquad (12)$$

Comparing researches between algorithm include computers' speed differences on which calculations have been executed. An algorithm HGA was tested on the computer with the Athlon (1,667 MHz) processor; GASA and TS algorithms was executed on an insignificantly slower computer with the Athlon (1,400 MHz) processor. Results presented in Table 6 show that solutions generated by the HGA algorithm are statistically better than results of the GASA algorithm. More precisely, for all 40 tested instances, the HGA algorithm generates better solutions in 39 instances. An average scheduled length obtained by the proposed HGA algorithm is 5% lower (better) than the scheduled length obtained by the GASA algorithm. Additionally, the HGA algorithm is about 2.5 times faster than GASA. Comparing with the next literature algorithm, TS [11], the proposed approach gives better results in 37 of the 40 instances tested. An average relative deviation to reference solutions was also lower (over 3%).

5 Summary

The presented methods of auto-tuning can be applied to any genetic algorithm. They are based on entities existing in every genetic algorithm regardless of the problem being solved. Automated adjustment of the mutation's probability of designating individuals which will undergo mutation and automatic scaling of the fitness function value takes place on-line that is during the run of an algorithm. The proposed choice of a crossover operator and mutation's operator takes place off-line that is in the initiating calculations. The beginning calculations can be carried one only once during the creation of the genetic algorithm dedicated to a given problem or built in the algorithm as an initiating phase carried out each time before solution to given instance. The values of the proposed solutions were verified on the genetic algorithm specialized to find the schedule of the job-shop problem with an additional "no-wait" constraint.

Acknowledgements The work was partially supported by the Polish Ministry of Science and Higher Education (W. Bożejko, grant no. N N514 232237; M. Makuchowski, statute grant no. 344 304).

References

1. Bożejko W, Makuchowski M (2009) A fast hybrid tabu search algorithm for the no-wait job shop problem. Comput Ind Eng 56:1502–1509
2. Framinan J, Schuster C (2006) An enhanced timetabling procedure for the nowait job shop problem: a complete local search approach. Comput Oper Res 33(5):1200–1213
3. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading
4. Graham RL, Lawler EL, Lenstra JK, Rinnooy KanA HG (1979) Optimization and approximation in deterministic sequencing and scheduling: survey. Ann Discrete Math 5: 287–326
5. Grabowski J, Pempera J (2000) Sequencing of jobs in some production system. Eur J Oper Res 125:535–550
6. Hall GN, Sriskandarajah C (1996) A survey of machine scheduling problems with blocking and no-wait in process. Oper Res 44:510–524
7. Holland JH (1992) Genetic algorithms. Sci Am 267:44–50
8. Lawrence S (1984) Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Technical report. Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania
9. Raaymakers W, Hoogeveen J (2000) Scheduling multipurpouse batch process industries with no-wait restrictions by simulated annealing. Eur J Oper Res 126:131–151
10. Reddi S, Ramamoorth C (1973) A scheduling problem. Oper Res Q 24:441–446

11. Schuster C (2006) No-wait job shop scheduling: tabu search and complexity of subproblems. Math Methods Oper Res 63(3):473–491
12. Schuster CJ, Framinan JM (2003) Approximative procedures for no-wait job-shop scheduling. Oper Res Lett 31:308–318
13. Beasley JE (1990) OR-library: distributing test problems by electronic mail. J Oper Res Soc 41:1069–1072 (1990). http://people.brunel.ac.uk/∼mastjjb/jeb/info.html
14. Ovacik I, Uzsoy R (2003) Decompisition methods for complex factory scheduling problems. Oper Res Lett 31: 308–318
15. Weinberger ED (1990) Correlated and uncorrelated fitness landscapes and how to tell the difference. Biol Cybern 63: 325–336
16. Wismer DA (1972) Solution of the flowshop scheduling-problem with no intermediate queues. Oper Res 20:689–697