FLOWSHOP OPTIMIZATION IN MATLAB - GENETIC ALGORITHMS APPROACH

Martin Žďánský, Jaroslav Poživil

Department of Computing and Control Engineering

Prague Institute if Chemical Technology

1 INTRODUCTION

The purpose of this paper is to show that methods of AI, genetic algorithms in particular, are very effective at solving difficult, important real-world problems, specifically the optimization of serial multiproduct batch plant sequencing, and to present results of our work in this field. This work deals with the problem of finding sequence of batches that minimizes the makespan, and discusses the application of different genetic algorithms to find such optimum sequence. We have created in Matlab a genetic algorithm, and set the algorithm's parameters, so that the result is well suited to the specific problem type while remaining flexible enough to be applicable to different problems in target group. This paper presents the analysis of performance of different algorithm configurations and parameter values.

While continuous processes are the ones most often encountered in large-scale manufacture of chemical commodities (e.g. sulphuric acid, sodium hydroxide), batch processes do have their place in the chemical industry. Batch processes are often found in manufacture of products such as specialty chemicals, pigments, pharmaceuticals, etc. The advantages of batch processes include high flexibility that allows the manufacturer to quickly react to changes in demands, to change technology based on current situation, and to quickly introduce new or modified products. Batch processes also allow easier sharing of some of the resources (e.g. production units, storage capacities, manpower). Two categories of chemical batch plants are widely recognized: multi-product plants and multi-purpose plants. In sub-category of serial multi-product plants, which compose the scope of this paper, the production line consists of a single set of processing units, the plant has only one path for all products, and this path consists of a chain of units where no branches or loops exist. Example of this topology is shown on Figure1.

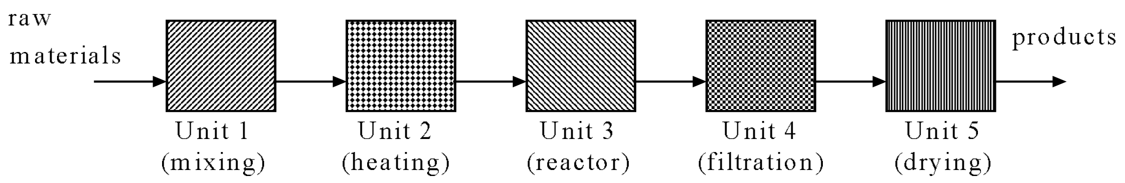


Figure 1. Example of flowshop configuration

Multiproduct plant operation can be described as a periodical manufacturing of many batches of distinct products in a series of campaigns. Efficient plant operation can be reached using different optimization criteria, and one of the most used ones is minimization of makespan. This requires that batches enter the flowshop in such an order that their passage through its chain of units is as close to „lock step“ as possible. Solving resulting multiproduct batch plant production scheduling problem consists of two sub-problems. The first one is the problem of determining start-times and completion-times for all operations, i.e. creating detailed schedule for a known sequence of batches. Results of this work are often displayed and used in the form of Gantt charts (Figure 2 shows simple example of such chart). The second sub-problem, hierarchically a higher one, is that of finding optimum sequence of batches. Finding optimum sequence of batches is an NP-complete problem.

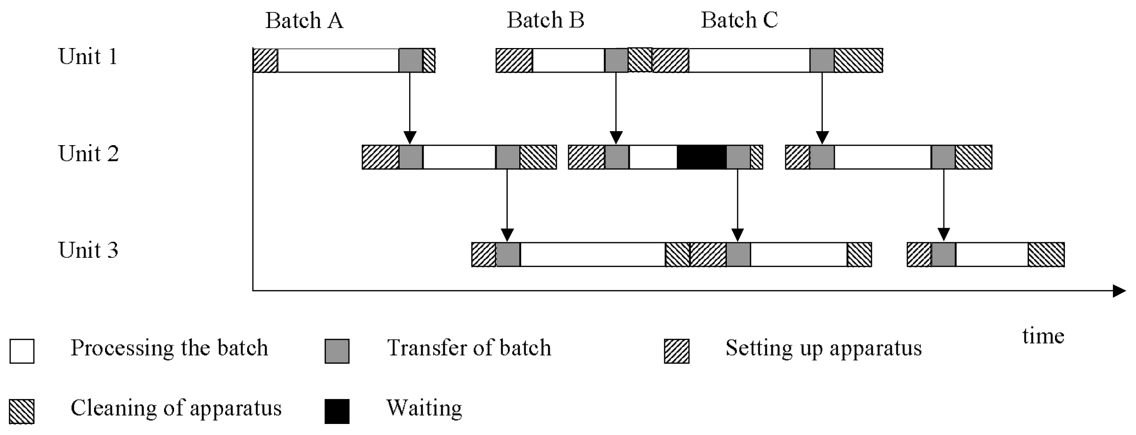


Figure 2. Example of a Gantt chart

Based on our results we can state that genetic algorithms (GAs) offer good performance in both solution quality and algorithm speed. Because many different variants of GAs exist, this work tries to find the ones best suited for solving flowshop sequencing problem, and to determine the influences of changes in different parameter values on the behavior of the algorithm and on the quality of the solutions found. The application of GAs to the problem is possible, as for used process models all information necessary to create schedule can be encoded in the form of a string of batch identifiers.

2 FORMULATION OF THE OPTIMIZATION PROBLEM

Batch processes can be modeled at many different levels of abstraction, and each of these levels is best suited for different purpose. The most important requirement is that the modeling must be economically effective and therefore the models should be selected and created to fit the requirements. Based on complexity, models are often divided into two basic categories: rigorous models and simplified models.

Rigorous models are based on exact formulations of physicochemical behavior. They are usually composed of systems of nonlinear algebraic and differential equations. The models are derived from basic laws of physics such as conservation of matter and energy, phase equilibriums and chemical equilibriums.

The simplified models are often based on time requirements of different operations occurring during manufacture. Based on the degree of simplification, different types of models of time requirements are recognized, but most of them are, thanks to their nature, flexible, applicable to description of most batch processes, because the amount of process-specific elements is zero or minimal. Serial multi-product batch flowshop model, as used here, is described by following input data and assumptions:

- a set of n batches manufactured
- a set of m units, arranged into a fixed sequence
- processing times matrix T, its elements t_{ij} corresponding to processing time for unit j and batch i
- transfer times matrix A, its elements a_{ij} corresponding to transfer time of batch i from preceding unit j-1

- set-up times matrix S, its elements $S_{(i-1)ij}$ corresponding to set-up time for empty apparatus j for production of batch i after batch $i-1$ (cleaning, adjusting); this value is sequence-dependent
- a unit may only process one batch at a time
- once started, a unit must complete the processing of a batch
- storage policy - algorithm was tested under four of the commonly used interstage storage policies: unlimited intermediate storage (UIS), finite intermediate storage (FIS), no intermediate storage (NIS), and zero wait (ZW)
- all processing, transfer and set-up times remain constant during considered interval

For illustration of the mechanism of the model, see Figure 2.

The objective criterion used in this paper is the minimization of the makespan, i.e., the time of completion of the last operation over the last batch in the sequence. Under abovementioned assumptions, if the sequence of batches is known then the completion times can be computed for all listed interstage storage policies. Resulting sequencing problems lead to NP-complete problems that cannot be solved by MILP, and require MINLP or other methods capable of solving non-linear mixed-integer problems.

3 GENETIC ALGORITHMS

Genetic algorithms are a general methodology for searching a discrete solution space in a way that is similar to process of natural selection procedure in biological systems. The algorithm is a remarkably general one, and it can be applied to different problems if following conditions are met:

a) solutions to the problem can be expressed in the form of a string of characters

b) a „fitness“ criterion, which in some way quantifies the quality of a solutions, can be computed for any valid string

c) strings in which „part“ of a good solution is present are rewarded by allocation of a higher fitness than „average“ strings

Genetic algorithms, as the name implies, are a type of algorithms, not a single one. This means that many variants of the basic idea exist, and that individual applications may be highly different. However, every variant should include following operations: (1) a method for encoding solutions to the problem into a string of characters; (2) an evaluation function which takes a string as an input and returns a fitness value which measures the quality of the solution the strings describes; (3) an adaptive plan, whose purpose is to produce new, improved generation of solutions from the current one.

GAs are often used in artificial intelligence applications, where the strings are often binary coded. This encoding, however, is not well suited for our purpose. Instead, the string is composed of a sequence of unique identifiers. Each identifier is represented by an integer number, and identifies a corresponding batch. The use of such an alphabet does not violate the principles of GAs. Encoding the solution in this way is enabled by the fact that the optimization is performed under the assumption of permutation schedules. This simplification means that the sequence batches are processed in is the same on all units, and that the whole schedule for a given input data set can be created from a permutation string that describes the sequence of batches. The strings containing substrings with small makespan generally have smaller total makespan compared to average strings, as required in c), allowing the use of GAs.

The evaluation function is based on the objective function, i.e. the computation of makespan. Considering that the ranges of processing times, as well as other values, can change for different applications, it is hardly possible to use the raw makespan value, and it must be somehow transformed to allow better algorithm function. Details on this are included later in this paper.

Adaptive plan, whose purpose is to produce new, improved population of solutions using information provided by the current generation, consists of the fitness proportionate reproduction, string crossover, and mutation operators. The probability that a particular string will participate in reproduction process is set proportional to its fitness. High fitness strings will have higher expected number of offspring. Crossover is the most important operator of a genetic-based technique. Different crossover operators have been proposed for a wide variety of problems. Basically, when two strings are selected for crossover, some of the characters they contain are semi-randomly exchanged. Crossover method must be chosen so as to guarantee that valid strings will be created. While crossover exploits already existing knowledge, new knowledge is introduced into the system by mutation. Basically, the mutation is the occasional and random alteration of some characters in a string. As with fitness evaluation, details on different approaches to adaptive plan and its parts are included in computation results section.

Compared to MILP and similar techniques, GAs offer much easier mathematical description, and many of the complications are not present because the optimization algorithm operates only on the strings containing the solution. The constraints of the problem, implicit in the assumptions listed in chapter 2, as well as all other details of a specific problem, are incorporated into function computing the makespan. As many different interstage strategies exist, each with its own makespan functions, it is easy to modify GA to use new interstage strategy, because the only part of the algorithm that has to be changed is the function computing the makespan. The algorithms presented in this paper used only the four interstage policies listed in chapter 2, but remains applicable, if some mixed or more complex interstage policies are used, as long as the three conditions listed at the top of this chapter are met.

4 APPLICATION OF GAS AND COMPUTATION RESULTS

In this section we describe the methods we have used in our work, with the results of test of different algorithm variants included in the relevant sections. Section 4.2 describes the new crossover operator we propose as well suited for flowshop scheduling problem, and following sections deal with our research of GAs built around this operator.

4.1 TEST PROBLEMS

Test problems were created by a random generation of input data matrices, as is often done in similar works. The range of processing times was 1-24, ranges of set-up and transfer times were 1-4, numbers of storage capacities under FIS policy were randomly generated in range of 0-1. The sizes of solved problems varied, and the results presented in this paper are mostly the ones for the problems with n={10,15,20} and m={5,10}. Every problem dimension was tested on 100 data sets.

4.2 CROSSOVER OPERATOR

We use a new variant of string crossover operator, based on crossover operator proposed by Cartwright and Long [1], and introduceded by Stluka [5]. In the original version, when two strings are selected for crossover, a segment containing the same number of batch identifiers, is selected at random from two strings and the segment is exchanged. For example, in the following two strings, the segments to be exchanged are shown in bold face:

String A	2	3	6	1	8	5	**4**	**10**	**9**	7

String B	8	4	**1**	**10**	**2**	7	6	9	3	5

This leads to two new strings:

String A	2	3	6	1	8	5	1	10	2	7
String B	8	4	4	10	9	7	6	9	3	5

in which duplicate batch identifiers appear, as marked by underlining. The two new strings are then analyzed from their first batch position. When a duplicate identifier is found, it is swapped with the first duplicate identifier in the second string,

String A	2	3	6	1	8	5	*4*	10	2	7
String B	8	4	*1*	10	9	7	6	9	3	5

and the process is continued until all duplicates have been removed.

String A’	2	3	6	1	8	5	4	10	9	7
String B’	8	4	1	10	9	7	6	2	3	5

Since the number of duplicates in the two strings must be identical, and batch identifiers missing from one string must appear twice in the second, this method always yields valid strings. However, we feel that this method is still too disruptive. The strings in the population are supposed to contain „parts“ of good solutions, and the described method tends to disrupt sequence of both the original string and the implanted segment, decreasing the quality of resulting strings and introducing unpredictable mutations into the algorithm.

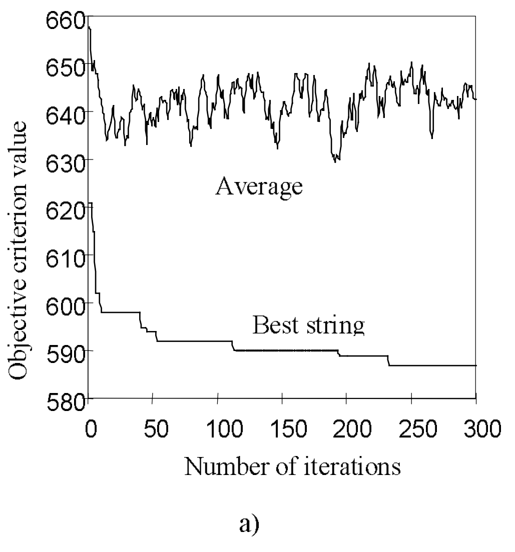
Our procedure starts with the same selection of two segments:

String A	2	3	6	1	8	5	**4**	**10**	**9**	7
String B	8	4	**1**	**10**	**2**	7	6	9	3	5

Then the first occurrence of the identifier located at the start of a segment from string A to be implanted into the string B is located in the string B, and then the sequence of identifiers selected from string A is inserted into string B into this position. To remove duplicate identifiers, the identifiers that appear in segment exchanged are deleted from their old positions in string B. The same operation is performed when implanting segment from string B into A, and this exchange leads to two following string:

String A’	3	6	**1**	**10**	**2**	8	5	4	9	7
String B’	8	**4**	**10**	**9**	1	2	7	6	3	5

Figures 3.a and 3.b illustrate the difference between two methods. The figures show the examples of results for typical optimization runs where crossover rate is set to 1 and mutation rate to 0. In the first case (Figure 3.a) the average makespan in the population remains approximately constant, as new information is introduced into the system by the crossover method in a way similar to mutation. The second figure shows the average makespan to rapidly decrease, because the disruption to the strings is much smaller and the population tends to move towards local optimum.



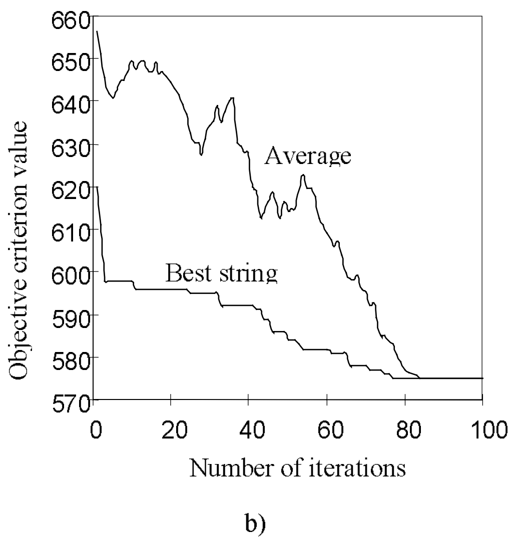


Figure 3. Objective criterion value of both best string and average string as a function of number of iterations in case of crossover rate 1 and mutation rate 0; a) crossover operator by Cartwright and Long [1] b) crossover operator as proposed by us

4.3 Adaptive Plan

Several other key parameters describe the adaptive plan. The parameters are: population size, reproduction operator, crossover rate, mutation rate, and number of iterations. They are discussed in following text.

4.4 Population Size

Generally, the greater the population is, the greater part of search space is covered and therefore better results can be achieved. On the other side, the computation time increases, and the convergence decreases. Figure 4 shows the average results for population size range 30-120. The mean relative deviation is somewhat decreasing, especially for larger problems, and the speed of the algorithm was great enough so that we decided to set the population near the higher end of this range, to 90 strings.

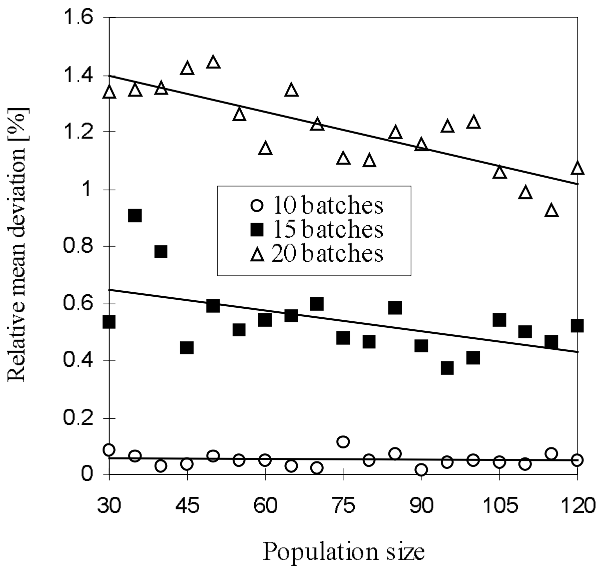


Figure 4. Relative mean deviation as a function of population size

4.5 REPRODUCTION OPERATOR

In some cases, it is possible to simply use the value of objective criterion as a fitness value. The character of the problem at hand does not permit this approach, and an appropriately modified value of objective criterion must be used. In this work, we use the linear fitness scaling method, where the fitness value $F(y)$ is computed from objective criterion value $C(y)$ (makespan)as follows:

$$F(y) = a.C(y) + b,$$

where a and b are constants. The constants can be computed in different ways, but the requirement that $F_{avg}=C_{avg}$ must be fulfilled. This guarantees that when proportional selection is used, average solution will be allocated exactly one position in reproduction pool. Maximum value of the new function is determined by equation $F_{max}=D.C_{avg}$, where D is desired number of copies of best solution string. The value of D is usually set to 1.2-2, but with relatively large population size our tests justified setting it to 2.6. Following equations can be derived for constants a and b:

$$a = \frac{C_{avg}(D-1)}{C_{\max} - C_{avg}},$$

$$b = C_{\mathbf{avg}} \frac{C_{\max} - DC_{\mathbf{avg}}}{C_{\max} - C_{\mathbf{avg}}}.$$

We have tested 5 different reproduction operators: deterministic, deterministic with single-string elitism, stochastic, stochastic with single-string elitism, random tournament. Use of deterministic reproduction, expanded by the addition of single-string elitism, was found to be most successful when average relative mean deviations were used to judge operator efficiency. The principle of single-string elitism gives better results for both deterministic and stochastic operators. Random tournament gave worst results and unless other research suggests otherwise we conclude its use should be discouraged.

4.6 CROSSOVER RATE

The relation between crossover rate and relative mean deviation was tested for range of crossover rate 0.1 - 1. The results of these tests are shown in Figure 5.1 and Figure 5.2. The first one shows that as the crossover rate increases the relative mean deviations slowly decrease. If the results of tests with problems of different sizes are merged, as shown in Figure 5.2, we can state that the best value of crossover rate parameter is approximately 0.85.

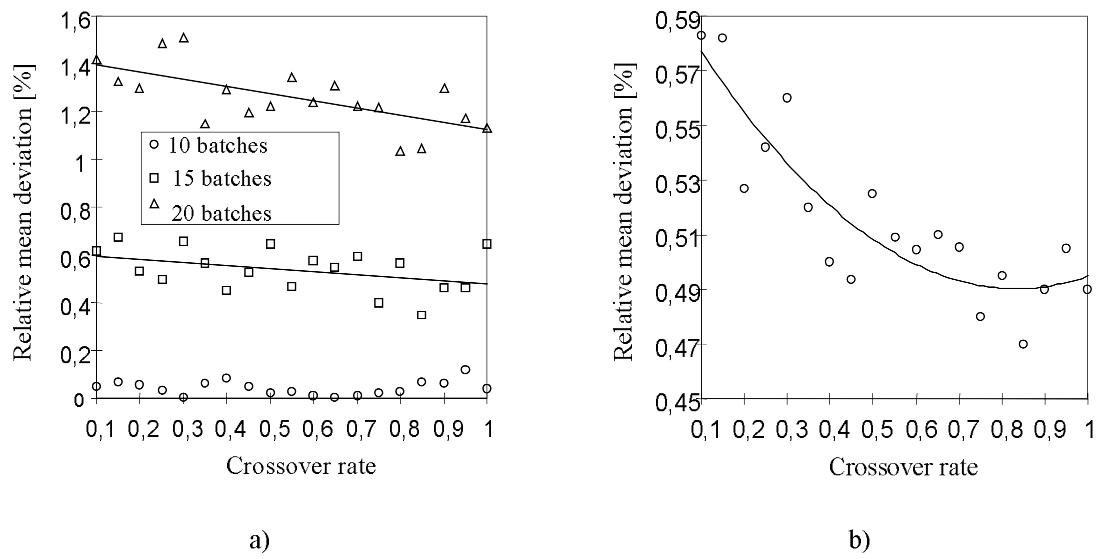


Figure 5. Relative mean deviation as a function of crossover rate; a) separately for different numbers of batches, b) average values for all problem dimensions combined

The length of segment to be exchanged was set to 3 characters. Our tests show that in cases of problems with 10 - 20 batches the optimum length of segment to be exchanged is approximately 0.2*n* and that relatively wide range of values around this one gives equally good results, and therefore the choice is justified.

4.7 Mutation and Mutation rate

In this work, the transform used to simulate mutation is the swap of two randomly chosen identifiers in a string. This method is fast, simple to code, and it gives very good results.

One of the important parameters of the GA is the mutation rate. We have tested range of mutation rate 0 - 0.1, and also use of dynamic mutation rate. The results show that even the best value of fixed mutation rate, i.e. 0.06, gives worse results compared to dynamic mutation rate. The principle of dynamic mutation rate as used by us are shown in Figure 6.

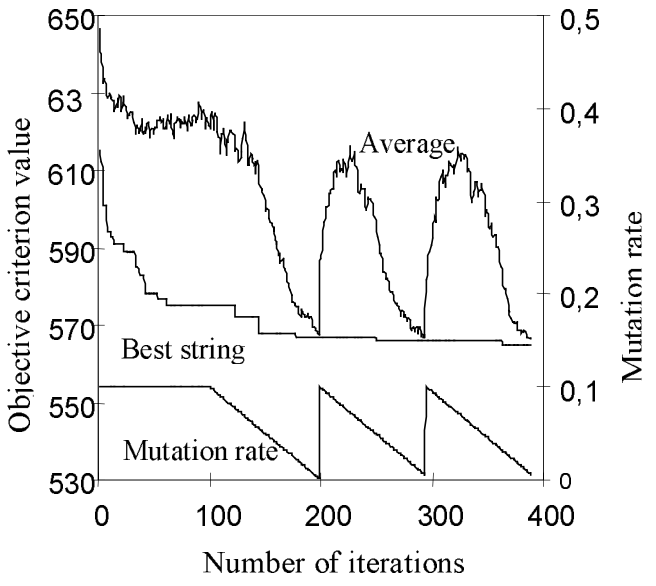


Figure 6. Principle of dynamic mutation rate

4.8 Number of Iterations

GA is not guaranteed to find an optimum solution, and it is not able to determine whether currently best solution is optimal one. For these reasons, the run of the optimization must be terminated on meeting some other criterion. With computation time limitations in mind, we have decided to end the optimization runs after 500 unsuccessful iterations. At this setting we obtained results approximately in seconds, and therefore it should be possible to use bigger value and to obtain better solutions of larger problems.

4.9 Solution optimality and Algorithm performance

As the algorithm is not guaranteed to find optimum solution, the question of solution optimality is to be addressed. The solution quality is in this work evaluated using relative mean deviation from optimum, but this optimum in most cases is not the global optimum. In the case of problems with 10 batches it is possible to find global optimum through enumeration (search space of 3 628 800 configurations), but for 15 and 20 batches the number of possible solutions is $1.31*10^{12}$ and $2.43*10^{18}$ respectively, making such approach impossible for today's computer hardware. As the research in this area tested other optimization methods beside GAs (tabu search, simulated annealing), the optimum solution for cases of n={15,20} was defined as the best solution to the given problem from all the solutions found using different methods and repeated optimization runs. Another indicator, search success rate, expresses the ability of the algorithm to find optimum solution (the optimum solution is defined as above). Table 1 shows summary results for the best found variants of GAs.

	10 batches	15 batches	20 batches
Relative mean deviation [%]	0.026	0.372	0.816
Search success rate [%]	89	33	4

Table 1. Genetic Algorithm efficiency

The table shows that although the GA cannot be guaranteed to find optimum solution, it does find good sub-optimum solutions. Moreover, the results shown in the table were obtained using termination criterion defined as 500 unsuccessful consecutive iterations. When this limit is raised, the algorithm is able to find better solutions, especially in the case of larger problems, and still finds such solutions in acceptable time. Another possibility for finding better solutions would be to use promising solutions from the final population as starting points for some local-optimization technique.

5 CONCLUSIONS

We have created in Matlab a genetic algorithm for solving scheduling problems. We have tested the possibilities of its application for sequencing of production in chemical batch plants, specifically in the case of the flowshop topology. We have found that Matlab realization of GAs allows us to obtain optimum or good sub-optimum solutions in acceptable computation times. The paper shows that genetic algorithms represent a very good method for solving such important problem as that of scheduling of batch operations.

Acknowledgements

This work has been supported by the Ministry of Education of the Czech Republic (program No. MSM 223400007).

References

1. Cartwright, H. M., Long, R. A., Simultaneous Optimization of Chemical Flowshop Sequencing and Topology Using Genetic Algorithms, Ind. Eng. Chem. Res. 32 (1993), pp. 2706-2713.
2. Androulakis, I. P., Venkatasubramanian, V., A genetic algorithmic framework for processes design and optimization, Computers chem. Engng 15 (1991), pp. 217-228.
3. Jung, J. H., Lee, H. H., Lee, I. B., A genetic algorithm for scheduling of multi-product batch processes, Computers chem. Engng 22 (1998), pp. 1725-1730.
4. Kim, M., Jung, J. H., Lee, I. B., Optimal scheduling of multiproduct batch processes for various intermediate storage policies, Ind. Chem. Engng 35 (1996), pp. 4058-4066.
5. Stluka, P. Využití prvků umělé inteligence při rozvrhování vsádkových výrob (Use of artificial intelligence for scheduling of batch operations) Ph. D. thesis, Department of Computer and Control Engineering, VŠCHT Praha 1998.

kontakt na autory:

Technická 1905, 166 28 Praha 6, Czech Republic

Email: martin.zdansky@seznam.cz, Jaroslav.Pozivil@vscht.cz

Tel.: +420 2 2435 4259, Fax: +420 2 2435 5053