SOLVING SCHEDULING IN PHARMACEUTICAL INDUSTRY WITH GENETIC ALGORITHMS

Elena Simona Nicoară

Petroleum-Gas University of Ploieşti, Informatics Department
Bd. Bucureşti, no. 39, Ploieşti, Romania, e-mail: snicoara@upg-ploiesti.ro

Abstract: *The paperwork presents results of applying genetic algorithms in Job Shop Scheduling Problems specific to pharmaceutical industry. These problems proved to be complex enough to require special approaches, from the demand forecast stage to the distribution stage. In this study there were applied, then compared, four genetic algorithms; there are two test cases: a real scheduling problem in pharmaceutical production and the test-instance JSSP-type ft10, both solved as uniobjective and multiobjective problems. The results show that the complex JSSPs can be efficiently solved using the NSGA_II algorithms or the elitist genetic algorithm, depending on the specific type of complexity.*

Keywords: *genetic algorithms, production systems, scheduling algorithms, multiobjective optimizations, performance evaluation*

1. STATE-OF-THE-ART IN PRODUCTION SCHEDULING

In production environments, one of the main purposes for global optimization is the scheduling of all involved tasks, so that both makespan and idleness time to be minimum.

The aim of this *paperwork* is to test genetic algorithms on production complex scheduling in order to detect the adequate conditions and the possible limitations associated to this process.

The terminology in scheduling theory refers to jobs formed of operations, resources, processing time for an operation on a resource, routings of jobs on the resources, production plant and schedules. The schedule represents an order for executing operations, plus the start processing times for each operation. The solutions in production scheduling are valid schedules.

The classes of production scheduling problems are multiple, based on the different production conditions:

- Minimum Job Shop Scheduling Problem (JSSP), where jobs are structurally heterogeneous;
- Minimum Open-Shop Scheduling, where the operations do not have to be processed in a certain order and the jobs are homogenous (each job is processed on each machine);

- Minimum Flow-Shop Scheduling, where the processing order on the machines is identical for each job.

The complexity of JSSP class is bigger, taking into account the heterogeneousness of the jobs and the possibility of alternative routings for an operation.

In this paperwork the focus is on this class of production scheduling.

1.1 Job Shop Scheduling Problem

The JSSP is placed among the most difficult combinatorial optimization problems. Formally, the JSSP is defined as it follows [8]:

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

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

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

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

Objective:
Minimization of makespan for the schedule, namely minimization of:

$$C_{\max} = \max_{i \in J} (f_m(o_{i,ni}) + \tau_{i,ni}) \quad (1)$$

If we note with $t_{i,j}$ the start processing time of operation j of job i, then we can rewrite the relation (1) as it follows:

$$C_{\max} = \max_{i \in J, 1 \le j \le n_i} (t_{i,j} + \tau_{i,j}) \quad (2)$$

The objective in JSSP is to find value of:

$$C^*_{\max} = \min(C_{\max}) \quad (3)$$

The JSSP can be viewed from many points of view. It can be deterministic or stochastic, static or dynamic, with flexible scheduling or not. The variability of the parameters indicates the level of incertitude. If this level is insignificant, the JSSP is deterministic. Otherwise, it is stochastic. In the static model of JSSP, the number and the characteristics of the jobs remain unmodified during the solving process, while in the dynamic model these vary. The scheduling is flexible if certain operations can be processed on more alternative machines, which form a set of alternatives.

The experience shows that JSSPs are not only NP-difficult, but very difficult to solve even heuristically [2].

For an accurate comparison of the techniques and algorithms designed to solve JSSP, there were created test problems, which provided a common standard for testing and comparing. These test problems represents, by dimension and structure of input data, certain scheduling characteristics. They are relevant for the industry, they are realistic containing attributes that produce commercial software products and research results, and their input data are easy to be read and interpreted. The instances have different dimensions and complexity levels; this fact leads to an easy determination of the abilities and the limitations of the methods tested on them.

The OR Library [1] provides over 80 scheduling test instances.

Among these, the ft10, the most famous, was proposed by Fisher & Thompson in 1963; it is a difficult one, the first solution being identified only after 24 years. It consists in 10 jobs, each being formed of 10 operations, to be scheduled on 10 machines.

Once that many schedules (different solutions) were determined, we can compare them by unary and binary performance measures (see figure 1).

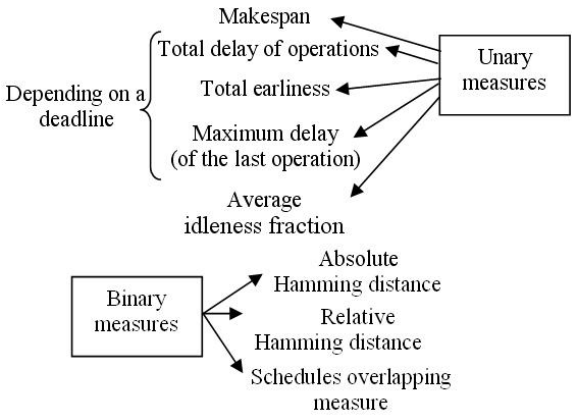

Fig. 1. Performance measures for schedules

The last unary indicator (average idleness fraction) is proposed by the author for measuring how compact the operations in the schedule sequence are. The more reduced its value is, the more compact the schedule is, because the waiting times of the operations are small.

The most of performance measures for schedules work at processing sequence level – they consider the operation order, but ignore the information about the start times associated to the schedule.

One measure that takes into account both the order of operations and the start and stop times is the schedules overlapping measure. This quantify the similarity of two schedules so that identical schedules (as order and times) will have a measure equal to 1, and two schedules completely different a measure equal to 0.

1.2 The specific of scheduling in pharmaceutical industry

The pharmaceutical industry requires special conditions, from the demand forecast to the final products distribution.

The hall-mark of job shop scheduling problems is their complexity, determined by the difficulty of representation, by the frequently hard constraints, by a big consume of resources for evaluating (partial) solutions and the huge dimension of the search space. The inherent restrictions of the production stage are:

- A high level of stringency, from the drug formula detection to the financial decisions;
- The diversity of conditioning and packing forms (sometimes, even for the same product) – tabloids, capsules, solutions, creams etc.;
- The often dynamic character of the operations; in other industries the processes remain constant for a certain product. The processing stages can depend on the previous or the next stages (ex. multiple levels of clean up for tools);
- Generally, intolerance to wait for the products, at any stage;
- The need to schedule so that certain stages must not be interrupted at a shift end (for those plants that do not work in flow);
- The impossibility to blend different substances in storing rooms;
- Strict rules concerning the quality. Frequently, the production is put on hold until the quality experts approve the next phases of processing;
- Flexible state of products: in formulation stage, in production, stored, medicine for approval or product in transit.

The strict quality standards are the main factor which places this industry on a distinct position against other production industries.

In every country there are professional agencies, which deal with legislation, foreknowledge and common practices in this field. In Romania, for example, these are the Health Ministry and the National Medicine Agency.

In a medicine production company, conformation to GMP norms (Good Manufacturing Practices) ensures the international standards. The GMP certificate is valid for both medicine fabrication and for division, packing and labelling. Other practices in the field are GLP (Good Laboratory Practices) and GCP (Good Clinical Practices).

Besides these norms is also favourable the international approval or expertise from certain governmental organisations (Food and Drug Administration, European Agency for the Evaluation of Medical Products or Environmental Protection Agency).

All these considerations imply supplementary requirements to the scheduling systems. These have to be flexible even if there are many constraints, to be able to represent and to solve problems for processes with diverse structures

and to generate detailed schedules for diverse periods (from one week to five years).

2. USING GENETIC ALGORITHMS IN PHARMACEUTICAL INDUSTRY

Specifically, the JSSP is a nondeterministic hard problem. Exact methods have been successfully applied in solving only small instances. Their most important disadvantages are:

- Lack of a global perspective;
- Inefficiency for problems with discrete variables;
- Tendency to block in local optimal regions;
- Convergence dependent on initial solution;
- Inefficiency for parallel work environment;
- An efficient algorithm for a specific problem may be inefficient for others problems.

In contrast, the approximate methods, though do not guarantee the global optimum identification, they find much faster almost optimal solutions, generally multiple. Among the approximate techniques applied in scheduling domain (GRASP - Greedy Randomized Adaptive Search Procedure, agent-based methods, genetic algorithms, PDRS - Priority Dispatch Rules Systems, simulated annealing and tabu search), the genetic algorithms prove to be well adequate to the complex and big problems, with no ordinary objective functions, such as scheduling problems in pharmaceutical production.

Moreover, the genetic algorithms, being evolutionary algorithms, present the advantage of simultaneous treatment for many candidate solutions, and by using the adequate mechanisms for maintaining the population diversity; they avoid blocking in local optima. Besides these advantages, we can add:

- Free derivative characteristics;
- The global perspective;
- Simplicity for the preparation of the optimization model;
- Parallel processing suitability.

The mathematical foundation for the evolutionary algorithms (and consequently for the genetic algorithms) is related to the evolutionary search algorithms. According to the theorems in evolutionary search algorithms area, if an instance of a search problem, based on fitness, can be described in terms of:

- a set of candidate solutions,
- a fitness function and
- a success fitness level,

then an evolutionary search algorithm can be described in terms of two functions (one of generating individuals and one of rejecting individuals), that maintain the state of current population [7].

As an effect, a JSSP can be solved with genetic algorithms if we hold a set of candidate solutions, an adequate fitness function and, eventually, a success level of fitness.

The first attempt to use genetic algorithms to solve JSSP belongs to Davis in 1985 [3]. He demonstrated that these algorithms are adequate to the simple scheduling, but the reasons for his poor results were the ad-hoc genetic operators, inadequate to the chosen encoding and the big memory consumer chromosome representation.
Over the last twenty years, the evolutionary algorithms have been applied to a variety of scheduling problems, those in production area being the majority.

The genetic encoding used is the most adequate for the JSSP - the permutation encoding; every candidate solution is a string of genes, representing the operations to be processed. One gene has the form (job_i, $operation_j$), meaning the j-th operation of the job i. An example of candidate solution (individual), is this:

(6,1)(76,1)(53,1) ... (69,3) ... (54,8)(60,8).

A candidate solution will be interpreted by the order of operations and the start times for every operation. In the tests these times are obtained by decoding the permutation in semiactive schedule.

The validity of a candidate solution is judged only from the precedence constraints point of view, because the decoding of a candidate solution is made so that the other constraints to be satisfied.

The fitness function used for candidate solution evaluation depends on the JSSP objective(s).

The test instances were solved:

- as uniobjective problems, where we aim minimization of C^*_{max} (see formula 2);
- as multiobjective problems, when we want minimization of every value in ($f_0(x)$, $f_1(x)$, $f_2(x)$), where:

x is the candidate solution;
$f_0(x)$ is the makespan: C^*_{max};
$f_1(x)$ is the number of delayed jobs;
$f_2(x)$ is the average idleness fraction(*idleFr*):

$$f_2(x) = \frac{\sum_{i=1}^{ns} \frac{dreal_i - dvirtual_i}{dvirtual_i}}{ns}. \quad (4)$$

Here, *ns* is the number of jobs, $dreal_i$ the actual duration of job i in the system and $dvirtual_i$ the duration that job i would spend in the system if all the needed resources were available when they are necessary.

A big average idleness fraction is translated by repeated waiting of some jobs in certain stages, a fact with implications on overloading spaces in the plant and on raising the number of delayed operations.

The genetic operators were applied in many variants, with the aim to identify the most adequate model for each operator. For selection we used the roulette-wheel model, the tournament model and the elitist model. The mutation variants were the frame-shift operator, the translocation and the inversion operator.

For crossover there were developed over 10 different operators for permutation encoding. Among these we used UX (Uniform Order-Based Crossover) and PPX (Precedence Preserving Crossover).

If the individual obtained by applying an operator is not valid, the next action is one of the following: apply an algorithm to legalize the individual, apply iteratively the operator until the result is valid or give up the operator application.

In the paper there were tested four genetic algorithms: the canonic genetic algorithm, one elitist genetic algorithm, NSGA_II and NSGA_II ADR (NSGA_II with Dynamic Application of genetic operators and partial population Reinitialization).
The canonic genetic algorithm requires replacing entire population at each generation with the new one, formed by applying genetic operators. This algorithm, designed by Holland in 1975 is:

1. t <- 0 (first generation)
2. pseudo-random initialization of population P_t
3. evaluate P_t
4. while evolution is not ended
    4.1. t <- t + 1
    4.2. selection in P_t
    4.3. crossover of parents selected
    4.4. insert the descendents in the new population P'_t
    4.5. mutation for P'_t
    4.6. evaluate P'_t
    4.7. P_t <- P'_t
5. return the best solutions in P_t

The elitist genetic algorithm uses an archive of best individuals, formed during the evolution; selection considers both current population and archive.

The classical NSGA_II (Non-dominated Sorting Genetic Algorithm), specially designed for multiobjective problems, sorts the population according to the level of non-domination, each solution being compared with each other solution to find if it is dominated. In this way there are constructed the non-dominated fronts (F) one by one, each consisting in individuals non-dominated by those in the subsequent fronts. NSGA_II uses as diversity preservation mechanism a crowding distance comparison operator, which guides the selection process towards the true Pareto-optimal front, by favouring the solutions in less dense regions in each front. This algorithm uses a binary tournament selection, where the selection criterion is based on this operator [4]. The pseudocode of NSGA_II is [4]:

1. t <- 0
2. pseudo-random initialisation of population P_t
3. quick sort (P_t)
4. Q_t <- new_population(P_t)
5. while evolution is not ended
    5.1. R_t <- $P_t \cup Q_t$
    5.2. F <- quick_sort(R_t)
    5.3. P_{t+1} <- $\varnothing$, i <- 0
    5.4. until $|P_{t+1}| + |F_i| \le N$
        5.4.1. P_{t+1} <- $P_{t+1} \cup F_i$
        5.4.2. crowding_distance(F_i)
        5.4.3. i <- i+1
    5.5. sor P_t t(F_i, $\ge_n$)
    5.6. P_{t+1} <- $P_{t+1} \cup F_i[1:(N-|P_{t+1}|)]$
    5.7. Q_{t+1} <- new_population(P_{t+1})
    5.8. t <- t+1
6. return the best solutions in P_t

The improved variant of NSGA_II is designed for multimodal problems, where the population

can get attracted to a local Pareto-optimal front. Although the crowded comparison operator ensures a good diversity, this is done along the current front; the lateral diversity is lost. To ensure a better convergence, the authors of [5] proposed a controlled elitism mechanism which controls the extent of exploitation. This is done by restricting adaptively the number of individuals in the current best non-dominated front using a reduction rate coefficient. This mechanism allows solutions from all non-dominated fronts to coexist in the population.

In this paperwork we applied the improved variant.

The NSGA_II ADR contains two mechanisms to avoid the premature convergence of the algorithm towards suboptimal regions: dynamic application of genetic operators and partial population reinitialization [6]. To treat the multiobjective aspect, in the canonic and the elitist algorithms, we aggregated the objectives by certain coefficients. For the NSGA_II and NSGA_II ADR, we used the Pareto dominance.

3. SIMULATION RESULTS

3.1 Test cases

The first test case is a deterministic predictive flexible JSSP, a real world problem in pharmaceutical industry; here, the lot production consists in fabrication of tabloids packed in blisters or bottles, both of them being then packed in boxes. The search space for this instance is very big; it contains approximately $26*10^{388}$ candidate solutions.

This test problem consists in 79 jobs of 16 different types, each having maximum 10 operations, processed on 20 machines. In addition, the routings involve the possibility to choose the best machine among many alternatives (the first available). The chromosome length is therefore 606 genes.

For this test case, the measure unit chosen for makespan is the number of 8 hours shifts.

The second test case is the ft10 test-instance. For this, the candidate solutions are formed only by 100 genes, but its complexity level is high enough. The best known solution has a value of 930 makespan.

From the multiobjective point of view, the deadline for the first test case is 44 shifts, and for the ft10 the deadline is the moment 1000.

3.2 Results

The comparative tests consist in execution of the four genetic algorithms for the two test cases, both solved as uniobjective and multiobjective problems. The presented results are obtained based on many sets of different parameters values. In addition, taking into account the random factor correspondent to the application of genetic algorithms, every algorithm was run many times for each set of parameters values.

We executed for every algorithm 50 runs for 5 different parameters values; the evolution was performed with a population of 300, 500 generations, crossover rates between 0.3 and 0.7 and mutation rates between 0.01 and 0.1.

The comparative results are presented in Table 1- 4; we use the following abbreviations: *BF* for the best fitness, *AvF* for average fitness, *WF* for the worst fitness, *DivSS* for diversity in schedules space (measured by the number of solutions identified per run), *DivOS* for diversity in objective space (measured by variance of objective values), *OSE1* for objective space exploration (measured by variation range dimension for makespan) and *OSE2* for objective space exploatation (measured by solutions with best makespan). *RT* specifies the run time.

For the uniobjective case the fitness is denoted by makespan value. For the multiobjective case:

- *BF* is the aggregated objective value plus the makespan associated with that value;
- *AvF* and *WF* are the average and, respectively, worst aggregated fitness;
- the coefficients used for aggregation were (0.5, 0.1, 0.4) for the first test case and (0.8, 0.2) for ft10.

Table 1: Performance measures obtained by the four genetic algorithms, in the first uniobjective test case

Algorithm Measure	Canonic	Elitist	NSGA_II	NSGA_II ADR
BF	58.25	48.56	48.16	**47.99**
AvF	60.37	49.02	48.64	**48.48**
WF	68.27	50.05	49.23	**48.99**
DivSS	1.40	67.40	184.20	**235.04**
DivOS	3.29	0.48	0.40	**0.33**
OSE1, shifts	10.02	1.49	1.70	**1.00**
OSE2, %	10%	23%	24%	**27%**
RT, s	19.08	14.24	**10.40**	10.80

These data show that the NSGA_II ADR is the best genetic algorithm viewed from all perspectives: solutions fitness, diversity in schedules space and objective space, exploration and exploitation of objective space. The NSGA_II algorithm obtains better results than the elitist, and the canonic genetic algorithm detaches from all of these by its low efficiency.

In the multiobjective first test case, the performance separation is not so clear any more (see Table 2).

Table 2: Performance measures obtained by the four genetic algorithms in the first multiobjective test case

Algorithm Measure		Canonic	Elitist	NSGA_II	NSGA_II ADR
	MinM	56.40	48.24	**48.08**	48.10
	Aggr.	43.76	29.97	**29.29**	29.66
	Solution				
BF	C^*_{max}	56.40	48.83	48.08	48.72
	delay Op	68.00	9.00	6.00	10.00
	idleFr	21.90	11.65	11.62	10.77
AvF		51.16	**29.30**	29.57	31.15
WF		58.00	**29.92**	**29.92**	34.08
DivSS		1.00	1.90	1.20	**15.66**
DivOS		5.06	0.27	**0.23**	0.70
OSE1, shifts		5.89	**3.55**	3.94	3.70
OSE2, %	C^*_{max}	39%	64%	62%	**78%**
	Aggr	12%	**32%**	28%	15%
RT, s		**30.95**	276.24	202.36	225.67

The NSGA_II algorithm obtained a solution with makespan 48.08 shifts, while the NSGA_II ADR obtained a makespan with 10 minutes bigger. Another aspect is that the last algorithm is able to identify in big proportion (78%) good quality solutions, especially from the first objective point of view, the most important.

Based on the Pareto optimal fronts, the dominance relation between the algorithms can be expressed as it follows:

- The elitist genetic algorithm, NSGA_II and NSGA_II ADR totally dominate the canonic genetic algorithm; it means that every solution obtained by the last one is dominated by a solution of others algorithms;
- NSGA_II and NSGA_II ADR totally dominate the elitist genetic algorithm;
- Between NSGA_II and NSGA_II ADR there is no dominance relation, meaning that they can not be compared based on the identified fronts.

By comparing the best makespans in the two tables, we see that the value obtained for the multiobjective case (48.08 shifts) is only 1.6 hours bigger than the value for the uniobjective case (47.99 shifts).

In all tables, bold values indicate best results.

For the ft10, the NSGA_II ADR algorithm is the best for the uniobjective case, obtaining solutions with makespan 1013 (see Table 3). This value is smaller (better) with 4% than the best makespan obtained by the elitist algorithm and with 16% than that obtained with NSGA_II algorithm.

Table 3: Performance measures obtained by the four genetic algorithms, in the uniobjective ft10

Algorithm Measure	Canonic	Elitist	NSGA_II	NSGA_II ADR
BF	1342	1054	1216	**1013**
AvF	1384	1213	1265	**1102**
WF	1553	1355	1345	**1306**
OSE1	211	301	**129**	293
RT, s	**0.11**	3.30	1.75	0.70

For the biobjective ft10 (see Table 4), the elitist genetic algorithm proves to be the most adequate. The NSGA_II ADR algorithm identifies the biggest number of solutions per run (8.36 in average), comparing to the elitist algorithm (4.3) and NSGA_II algorithm (5.5).

This indicates a better diversity of the solutions in the schedules space. Regarding the diversity in the first objective space, the biggest dimension for the makespan variation range is obtained when the elitist algorithm is used.

Table 4: Performance measures obtained by the four genetic algorithms, in the biobjective ft10

Algorithm / Measure		Elitist	NSGA_II	NSGA_II ADR
	MinM	**1054**	1175	1096
	Aggr.	**844.2**	943.2	878.4
BF	*Solution*			
	C^*_{max}	1054	1175	1096
	delayOp	5	16	8
AvF		**823.5**	934.6	890.3
WF		**893.8**	992.1	956.5
DivSS		4.3	5.5	**8.36**
OSE1		**412**	456	213

The simulation results were obtained on a PC with AMD Athlon 1600 MHz processor and 256 MB RAM.

4. CONCLUSIONS

In the attempt to test genetic algorithm usage in solving deterministic flexible JSSP, we applied four genetic algorithms on two test cases, seen as uniobjective and multiobjective problems. The algorithms are the canonic genetic algorithm, an elitist genetic algorithm, the NSGA_II and the NSGA_II ADR algorithms.

The results obtained point out the suitability of the last three algorithms for the complex JSSP. For the first uniobjective test case, the real deterministic static flexible JSSP, the best makespan obtained by using genetic algorithms is 9% better than the makespan obtained through empirical methods. In the multiobjective case, the makespan is bigger than 47.99 with only 1.6 hours.

The performance values obtained for the second test case, ft10 test-instance, also show that genetic algorithms are recommended for this kind of problems. The best makespan obtained is 1013. Even if this value is bigger than the best known value (with 0.089%), many other applied methods obtained much poorer values than this. As a final conclusion, the results are satisfactory for the performance criteria of evolutionary algorithms: covering and diversity of search space and objective space, quality of the solutions and algorithm convergence.

REFERENCES

[1] Beasley, J.E., OR library: distributing test problems by electronic mail, European Journal of Operational Research 41, p. 1069, 1990.

[2] Brucker, P., Knust, S., Complexity results for scheduling problems, Osnabruck University.

[3] Davis, L., Job shop scheduling with genetic algorithms, Proceedings of the International Conference on Genetic Algorithms and their Applications, San Mateo, Morgan Kaufmann, p. 136, 1985.

[4] Deb, K., Agrawal, S., Pratap, A., Meyarivan, T., A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: IEEE Transactions on Evolutionary Computation, 6(2), p. 182 2000.

[5] K. Deb, T. Goel (2000), Controlled Elitist Non-dominated Sorting Genetic Algorithms for Better Convergence, Lecture Notes in Computer Science, vol. 1993, Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization, Springer-Verlag, London, p. 67, 2001.

[6] Nicoară, E.S., Contributions regarding evolutionary algorithms usage in pharmaceutical production precess, doctoral thesis, Petroleum-Gas University of Ploieşti, 2005.

[7] Sharpe, O.J., Towards a rational methodology for using evolutionary search algorithms, doctoral thesis, School of Cognitive and Computing Sciences, University of Sussex, 2000.

[8] ***, A compendium of NP optimization problems, in Complexity and approximation combinatorial optimization problems and their approximability properties, by Ausiello et al., Springer Verlag, 2004.