Procedural Optimization Models for Multiobjective Flexible JSSP

Elena Simona NICOARA
Petroleum-Gas University of Ploieşti, Romania
snicoara@upg-ploiesti.ro

The most challenging issues related to manufacturing efficiency occur if the jobs to be scheduled are structurally different, if these jobs allow flexible routings on the equipments and multiple objectives are required. This framework, called Multi-objective Flexible Job Shop Scheduling Problems (MOFJSSP), applicable to many real processes, has been less reported in the literature than the JSSP framework, which has been extensively formalized, modeled and analyzed from many perspectives. The MOFJSSP lie, as many other NP-hard problems, in a tedious place where the vast optimization theory meets the real world context. The paper brings to discussion the most optimization models suited to MOFJSSP and analyzes in detail the genetic algorithms and agent-based models as the most appropriate procedural models.
Keywords: *JSSP, Multiobjective, Optimization, Genetic Algorithm, Agent-Based Model*

1 Introduction

Optimization is a requirement for a very wide spectrum of real world applications. Regarding the theory of optimization, this is very well developed in certain areas (such as linear programming and, generally speaking, exact methods), but still an open research topic in other areas (as heuristic approaches). Whatever the theoretical method chosen to solve a certain class of real problem, an optimization model for that context is initially needed. For many problems, it is implicitly included in the optimization method and practitioners do not handle it as a separate stage in problem solving.

An *optimization model* has three components: variables, objective(s) and constraints and, based on these and problem specific input data it generates as output optimal values for the variables and the associated objective value(s). In other words, an optimization model recommends actions to obtain the best solution(s); it is an optimization prescription.

Optimization models have many advantages and limits as well. We could mention here a more or less rigidity in model the reality, an inherent simplification of reality (selection over all the actual interacting factors), difficulties in best specification the objective function, partial parameter accuracy, omitting delays in the complex systems, biases of the modeler, time pressure constraints, model simplifying, assumption adopted when approaching the complex systems [1], [2]. All these emerge when the optimization theory meet the real world context. A very good support in this direction is [3], where an engineering point of view of optimization theory is presented. A possible solution could be the usage of a two level repetitive simulation to obtain suboptimal but feasible solutions [4].

In *manufacturing*, the most critical optimization aspect is time efficiency. Around this concept, for the manifold production contexts various scheduling problems frameworks were designed: flow shop scheduling, job shop scheduling, open shop scheduling [5], [6], [7].

A *Job Shop Scheduling Problem* (JSSP) states that a finite set of heterogeneous jobs composed by many operations have to be optimally scheduled on a set of finite machines (resources) such that the precedence constraint, the non-preemption constraint and the resource capacity constraint are satisfied. This means that operations of every job must be processed in a predetermined order, every operation must not be interrupted and a machine processes only an operation at a time. The objective is to minimize the make span for the entire set of jobs. The output of the JSSP is therefore a time-optimal allocation of the limited machines to the operations of jobs, named optimal schedule. The most part of theoretical and practical background in

JSSP concerns the non-flexible uniobjective condition [8], [9], [10].
If moreover the routes of the jobs on the resources are flexible, or the structure of the jobs varies, or the resource set varies during the scheduling, *Flexible JSSP* (FJSSP) is the right framework to use [11].
The scheduling process becomes more complex when, additionally, multiple objectives have to be simultaneously satisfied, for example: maximize workload, minimize lateness, minimize jobs flow time, minimize work-in-process, minimize cost to set the machines, maximize total workload on the machines and so on. This is what is called *Multiobjective FJSSP* (MOFJSSP). A mathematical formulation for the MOFJSSP, as an extension for the deterministic predictive JSSP (presented in [8]) to include multiobjectiveness and alternative routings for the jobs is made in [12].
Scheduling in many industries is based on the MOFJSSP production system. We could mention pharmaceutical industry, chemicals, food industry, furniture, electronic devices and so on. For all these manufacturing processes, various optimization models are available, and all of them can be framed in the so-called discrete-event system model (DES).
To build a mathematical optimization model which contains all the tangible and intangible factors which determines the evolution in time of a DES is an ideal aim, but such a model is overly complex: it would comprise hundreds or even thousands of variables and would not allow analytical solutions [13]. Consequently, most of the research was focused on tools of other type to represent, to model and to simulate DES. These optimization models are various. Some of them are so-called *conventional models,* because they are built on the process model, which constitutes the core concept in the optimization model. In the (MOF) JSSP context, the adequate conventional models are: Petri nets, waiting systems, general decision models, logical formulations (such as STRIPS language), Markov processes and Monte Carlo simulation. In [14] and [15] an analysis of these models for (MOF)JSSP was made: the limits and advantages for all the models were pointed out, and the particular characteristics of the manufacture processes that require certain models as the most adequate were noticed. The main conclusion concerning the conventional optimization models is that they are more suitable for small and middle-size scheduling processes.
The other optimization models are named *unconventional models*; they place the process model on a secondary layer and the primary role in modeling is assessed here to a procedure that controls the system. This procedure may be represented as a sequence of instructions, most likely to transpose in algorithms. Therefore, the unconventional models are also named *procedural models*, and their great development in the last decades was sustained by the mentioned major difficulty in rigorous mathematical characterization of big complex technological processes behavior; in this case, it is not possible an approach which easily covers all the possible states of the system [13]. Most of the procedural models use techniques and mechanisms specific to the biologic systems, especially representation schemata and generation of behaviors, and for that reason they are included in the artificial intelligence field. By the help provided by this kind of models, we try to replace the human operator which has a "behavior based on skills" with applications used as intelligent assistants to facilitate good decisions in less time [16]. Among the procedural optimization models significant are: evolutionary algorithms in general and genetic algorithms in particular, agent-based models (negotiation techniques, Ant Colony Optimization, Particle Swarm Optimization, Wasp Behavior Model, artificial bee colony algorithm), neural networks, fuzzy techniques, expert systems and knowledge-based systems. The last four models were detailed in [15].
In the section 2 and 3 of the paper a broad analysis of genetic algorithms and agent-based systems as procedural optimization models for MOFJSSP is conducted, and section 4 concludes the paper.

2 Genetic Algorithms for MOFJSSP

Genetic algorithms (GA) belong to the sub-symbolic paradigm of artificial intelligence theory, where intelligence is the result of collective interaction of a great number of simple entities which work independently, in parallel, continuously and interconnected. Knowledge is implicitly represented, in a diffuse manner. Intelligence emerges by endogenous adaptation, apparently in a random way, provided that enough trials were achieved, and it cannot be "caught" in a symbolic form. GAs do not learn by cumulating knowledge, but by modifying the global structure of the model; in fact, GAs are more likely trained than programmed [16].

A GA is a parallel algorithm which transforms a population of mathematical objects (possible solutions of a problem called individuals) in a new population using three operators similar to evolutionary process in nature: proportionate reproduction based on fitness („best survives" principle), sexual genetic crossover and random mutation (genotype alteration) rarely applied.

Evolution starts from a random or pseudo-random population and occur in hundreds or thousands generations. In every generation, the steps are: population fitness evaluation, selection of some individuals, crossover and mutation for the selected individuals. The new population obtained in a generation becomes current to the next iteration of the algorithm.

The GA optimization power consists in a bias for population performance to grow after many generations, similarly to the natural evolution. This is determined by the competition between individuals for the resources and by the genetic material propagation from the best individuals to the next generations. The evolution achieved by a GA can be viewed as a travel in search space, on many ways, in many generations, towards regions with better performance.

The pseudo-code of generic GA is as follows:

```
1. $t$ <- 0 (first generation)
2. pseudo-random initialization of the initial 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 main varying operator in GA is crossover; by this operator, the genetic material of two individuals called parents is combined to produce offspring which inherit their characteristics. Another varying operator is mutation, which brings new genetic material in population, supplying the crossover action, which regularly cannot do this [9]. The selection favors survival of the fittest. Figure 1 shows the genetic operators interaction in a generation of GA.

A manufacturing scheduling process can be modeled with GA simply following the GA framework: simulate evolution for a population of abstract representations (called *chromosomes*) of candidate-solutions (*individuals*) towards better solutions. The candidate-solutions here are schedules. A chromosome corresponds to a genotype that belongs to the genome associated to the problem to solve. It is decoded by a particular mechanism to express the candidate-solution, whose performance, called *fitness,* may be tested (phenotype of that problem).

A genotype space for a JSSP, related to the corresponding space of individuals and to the phenotype space is depicted in Figure 2. Here, the elements of the genome are of (*a,b*)-type, where *a* and *b* are non-negative integer values dependent on the JSSP in-

stance.
The space of individuals corresponds to the decision space (the search space) and the phenotype space corresponds to the objective space in the optimization theory.
A GA does not identify all the feasible individuals in the search space, but evolves chromosomes with good phenotype. In JSSP context, a GA does not search all the valid schedules in the schedule space to evaluate them, but evolves mathematical representations of schedules (as numeric sequences (a_1,b_1) (a_2,b_2)… (a_n,b_n)) with low makespan. A statistician would need samples from billions of regions in the search space to efficiently estimate the quality of the space of individuals, while the GA reach the same result with much lower number of strings and no calculus at all [17].

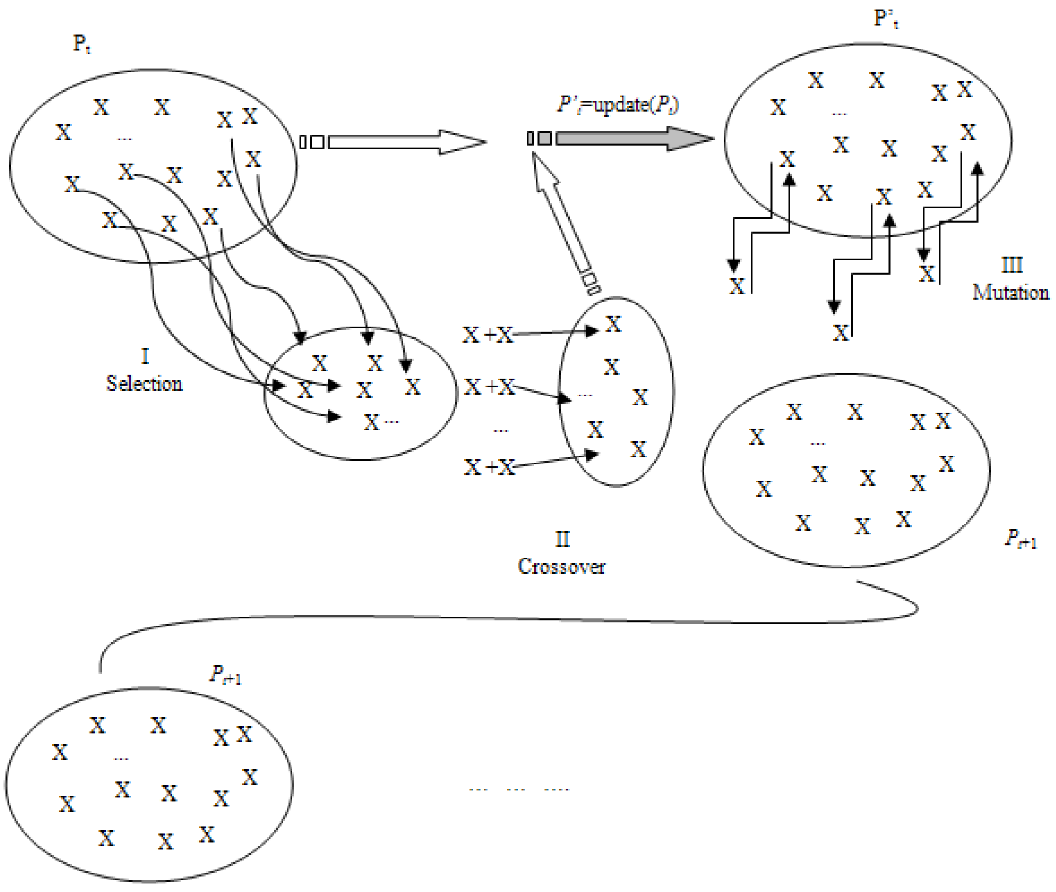

Fig. 1. Genetic operators in a generation of GA [12]

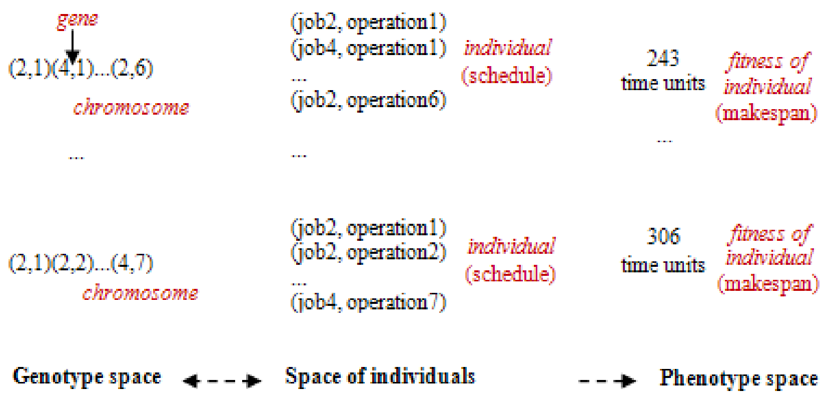

Fig. 2. Genotype space example for JSSP, in correspondence with the space of individuals and the phenotype space

The first attempt to apply GA in scheduling belongs to Davis [18]. Later, many studies were conducted by many researchers (to name a few: [7], [8], [10], [11], [12], [16], [19], [21]). Study topics regard genetic representation, fitness schemata, genetic operators, special mechanisms, hybridization of GA etc. Most research and applications concerned uni and multiobjective JSSP. The FJSSP subject was less handled, being a more complex process [4], [7], [10], [11], [12], [21], [22].

Investigation regarding genetic representation for JSSPs is pretty vast. In [19] a binary representation is proposed: to each pair of operations (o_1, o_2) which are processed on the same machine a binary variable is associated (1 if o_1 precedes o_2 and 0 if o_2 precedes o_1 in the considered candidate-solution). This representation allows using simple genetic operators, but repairing unfeasible new individuals is necessary and the representation dimension is relatively big.

In [23] the similarity of JSSP with traveling salesman problem is exploited and therefore a schedule is represented as a sequence of numbers $\{1, ..., n\}$, where n is the number of operations to be scheduled. A sequence is decoded from left to right and the numbers is used as indexes in a circular list of unprocessed operations.

Another representation, by scheduling graph, is proposed in [24].

The most common genetic encodings still remain permutation of set of operations to be scheduled [20] and the more direct representations based on start times and end times of operations, used as priorities in decoding. The later ones can not represent unfeasible schedules, but are regularly redundant – a schedule can be represented in many ways; certain schedules have a small number of representations, others a huge number.

The representation by permutation without repetition for FJSSP is extended in [22] by encoding an operation as (*job*, *operation, k*). The procedure to decode the chromosome in a schedule sequences the operations (*job*, *operation*) on the machine *k* in accordance with machine and operation availability times.

A feasible chromosome will be interpreted by:

- processing order of operations and
- start processing times for every operation.

Generally, the genetic encoding stores only the order of operations in the candidate-solution. The start processing times for each operation (which must be added to that order to obtain a schedule) are computed by decoding the sequence of operations in semi-active, active or non-delay schedule [11].

An interesting representation for the schedule is as sequence of rules (for example priority rules which selects operations in the set of operations to be scheduled) [7].

Once chosen the genetic encoding for the GA-based optimization model, adequate crossover and mutation operators must be designed or selected from the many proposed in the literature:

Order Crossover, Uniform Order-Based Crossover, Precedence Preservative Crossover, Subsequence Exchange Crossover, Partially Mapped Crossover, Time Horizon Exchange, Order Based Mutation, Swap Based Mutation, *frame-shift*, *translocation, inversion* and so on.

Selection does not interfere with the genotype, and therefore is problem independent. One can use roulette-wheel selection, tournament selection, elitist selection or other types of selection.

To evaluate the candidate-solutions the objective function(s) are used. In the uniobjective JSSP, fitness is makespan. For the multiobjective JSSP three methods to evaluate candidate-solutions are available:

- weighted aggregation of the objectives in one single objective;
- alternating the objectives with constraints;
- Pareto dominance relations.

Choosing GA to model a MOFJSSP problem calls forth all the benefits associated to GAs: allowing to obtain multiple different solutions (because GA work simultaneously with multiple candidate-solutions), allowing parallel processing, low cost for development, easiness to design, to implement, to extend and to combine with other optimizers. Some optimization methods tested in hybridization with GA are: local search, shifting bottleneck heuristic, beam search, simulated annealing, neuronal techniques, fuzzy logic, tabu search, priority dispatch rules systems, clustering algorithms and agent-based methods [7], [10], [12].

3 Agent-Based Models for (MOF) JSSP

Generally speaking, the agent-based technology, also called multi-agent technology, is underlain by a self-organizing collective system of interacting agents, which communicate on a cooperative or competitive basis. The main characteristic of an agent-based model is that an even low communication between agents leads to a complex and coordinate group behavior directed to accomplish a specific goal. These models mimic natural societies and are also studied as artificial intelligence methods.

The multi-agent idea proved to be attractive to model also manufacturing scheduling processes because the main entities of a manufacturing system (machines, workers, batches, manufacturing line) can be viewed as autonomous agents able to communicate with other agents in order to accomplish the production plan [7], [25], [26]. Some of the research in the area focused on applying to scheduling the multi-agent techniques tested before on other problems and the results were mostly satisfying. This is the case for Ant Colony Optimization, Particle Swarm Optimization, artificial bee colony algorithm and negotiation-based techniques. Wasp Behavior Model, on the other hand, is an example of agent-based technique suited and tested only for JSSPs.

Regardless the technique, the model generally operates with two types of agents: job-agents and machine-agents, which communicate by method-specific messages. A classifying structure for the multi-agent systems in manufacturing scheduling is proposed in Figure 3.

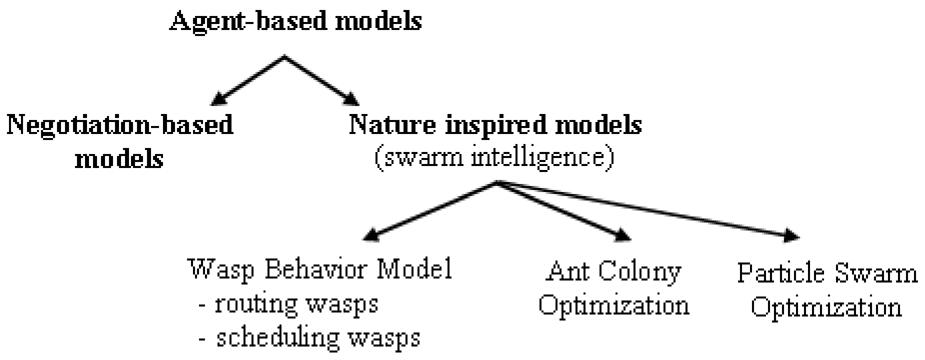

Fig. 3. Agent-based models applied in manufacturing scheduling

In **negotiation-based models** (or **market systems**), the agents interact by mutually acceptable „agreements” in order to accomplish the scheduling objective(s). For a negotiation to take place a sequence of bids and counter-bids concerning ready to apply changes in the system is needed.

Such a system for FJSSP is described in [7]. Initially, every job-agent receives a time budget depending on certain factors: the importance of that job in the production plan, its deadline and its processing time. When an operation of the job is ready to be processed, the job launches a request message to the adequate machines, specifying a time interval to end the operation processing. Every available machine-agent transmits a bid message with time interval when processing is possible and the corresponding price. The goal of machine-agents is to maximize own profit, while the goal of job-agents is to process the operation in the specified time interval at a minimum cost (generally depending on the cost of next operations). The job-agent compares all the received bids, if they exist, and choose the best one. If no bid is received, it

has to relax the request terms; more specifically, it has to modify the time interval for processing. Setting the optimal decision rules, both for job-agents and machine-agents, is not an easy task, especially when flexibility and multiobjective condition are present. Same is valid for handling the resource conflict when a machine-agent receives requests from more than one job-agent for the same time interval.
Another market systems type approach, a general one, focuses on a negotiation where the goal of every agent is to enhance the own value. This value strongly depends on the level of objective accomplishment. Regularly, the measure is in [0,1]; value 1 means that the agent accomplished all the objectives without violating any constraint and value 0 means that the agent accomplished no objective [7].
In the **nature inspired models**, unified under the **swarm intelligence** term, the group of agents is called colony. Here, the core concept is the emerging colony intelligence which efficiently solves optimization problems: an individual agent is simple, even primitive, and has limited perception and action capabilities in the colony, but the interactions between agents, even local, are those who lead to an efficient, adaptive, flexible, robust to noise global behavior; and all these happen without a central planning.
In ***Wasp Behavior Model*** (WBM), proposed by Theraulaz et al. [27], the task allocation to agents originates from wasp behavior, where every wasp has a response threshold for each region of the nest that influences the offspring feeding process. The social hierarchy in the colony leads to a dynamic distribution of tasks to the members, and the stimulus-response mechanisms in the model produce interactions between the colony members and between members and the local environment.
The WBM methodology was proposed first time for manufacturing scheduling and most of the research handled JSSP [25], [27], [28]. The agents are associated either to machines or to jobs. Therefore, two WBM models are used:

- *routing wasps model* and
- *scheduling wasps model.*

Figures 4a and 4b, adapted from [25] and respectively [28] clearly depict these models.

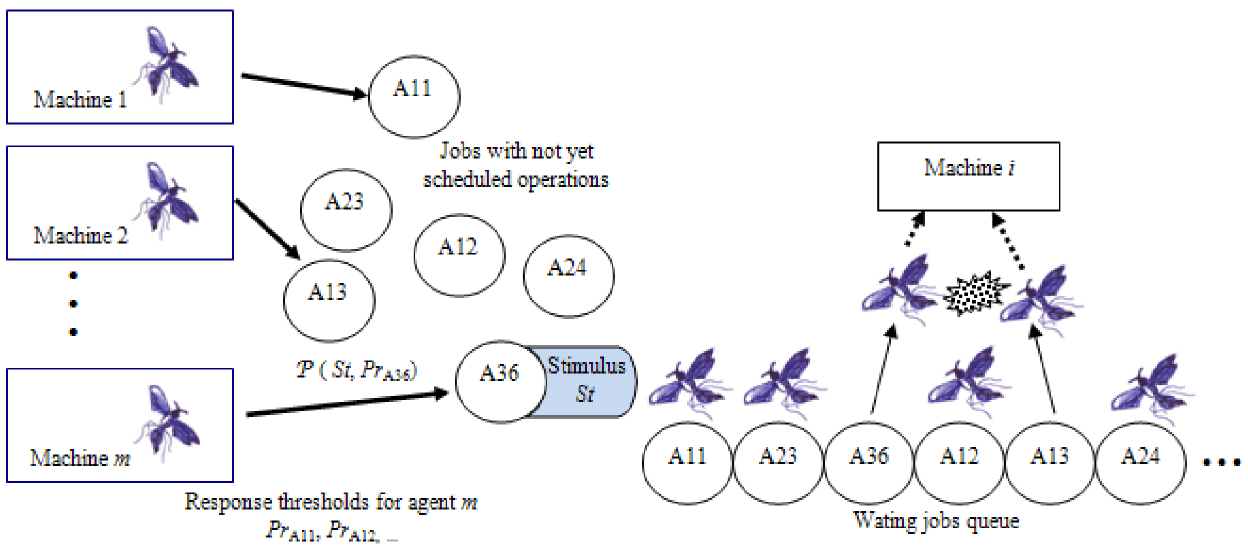
Machine 1
Machine 2
Machine m
A11
Jobs with not yet scheduled operations
A23
A12
A24
A13
$\mathcal{P}(St, Pr_{A36})$
A36
Stimulus St
Response thresholds for agent m
$Pr_{A11}, Pr_{A12}, ...$
Machine i
A11
A23
A36
A12
A13
A24
Wating jobs queue

Fig. 4. WBM models for JSSP
(a) routing wasps; (b) scheduling wasps [12]

In the first model, every machine has associated an agent in order to assign to its queue jobs to process. Every job with not yet scheduled operations emits a stimulus based on processing times specific to first operation to be scheduled; an agent chooses such a job to process probabilistically, based on its own set of response thresholds and on stimulus level of that job. Every agent knows in any moment its queue status and uses this information to adjust the response thresholds for every type of job [25]. If scheduling flexibility is considered, where an operation can be processed by a machine in a set of alternatives, the response thresholds for all the agents in that set are accordingly settled, and therefore the probability to choose that job is higher for all these agents.

In the scheduling wasps model, every job in the waiting queue is an agent in the system, named scheduling wasp, and compete with other agents in order to undertake the corresponding machine. The contest result is determined by the force values of the two competing agents, and these values depend on the machine setting times and processing times of the operations already scheduled or to be scheduled. The results of all the contests determine the jobs priorities, which further determine the schedule. The sequence of agents pairs to compete, when the needed machine becomes available, is decided by a type of the tour, chosen at the beginning of the modeling, as presented in [28]. If FJSSP is the case, the contest rules become more complex, as an agent will compete with more other agents corresponding to all the alternatives machines.

Whatever model may be, the solution for JSSP (the optimal schedule) is obtained iteratively, based on probabilistic decisions of the agents, until all the operations are scheduled. For the multiobjective case, no major alteration of the model is needed.

Ant Colony Optimization (ACO), proposed by Marco Dorigo [26], is a metaheuristic successfully applied to combinatorial optimization, simulating the group behavior of ants, which have the ability to find the shortest path to the food source by pheromone communication. To solve such instances a specific-to-problem graph of components (nodes) is built, where every node has a pheromone value associated. In every generation of the ACO procedure the agents, called artificial ants, visit the graph in order to build solutions, which are consequently sequences of components. Generally, for every visited node the pheromone value updates; the pheromone quantity accumulated on every covered route being proportional with frequency of covering, it guides the colony to identify the optimal solution (the optimal route in graph).

ACO is similar to genetic algorithms by maintaining in each iteration (generation) a population of candidate-solutions, but the distinction is made by the different generation mechanism of these candidate-solutions.

Specific to the manufacturing scheduling process, the graph components are the operations to be scheduled and the solutions built by the agents are paths in that graph - valid sequences of operations (schedules). An agent builds such sequence step by step, starting from a random valid operation and exit the graph when all the operations were visited. The solution building process is a probabilistic one and takes into consideration the pheromone trails on the nodes, which are deposited from the first iteration. The pheromone reflects the experience of the agents in searching the optimal schedule. Every agent proceeds in the graph selecting in the feasible region:

- with probability α the node with maximum pheromone;
- with probability 1-α a random node.

The α parameter translates the natural tendency of the ants to probabilistically choose a direction in search for food; they mostly prefer the routes marked with much pheromone, namely the routes chosen by other ants in the past. This cooperative interaction between ants is the guiding mechanism toward the shortest route.

At every step, once selected a node, the pheromone for that visited node is updated by a particular fixed or variable value. The update is proportional with the partial solution quality and inverse proportional with pheromone evaporation rate (if the procedure includes this mechanism to avoid a fast convergence to suboptimal regions of search space) [24]. In (F)JSSP the partial solution quality is the schedule makespan. In the

MO(F)JSSP the partial solution quality takes into consideration all the objectives.
In the following, an example of using ACO for a (MOF)JSSP with at least 60 jobs and at least 540 operations to be scheduled is graphically described. Two partial (valid) schedules built by two agents in the corresponding graph are:

- $o_{27,1}$ - $o_{5,1}$ - $o_{5,2}$ - $o_{60,1}$ - $o_{59,1}$ - $o_{5,3}$ - $o_{38,1}$ - $o_{60,2}$ and
- $o_{60,1}$ - $o_{60,2}$ - $o_{59,1}$ - $o_{5,1}$ - $o_{38,1}$ - $o_{5,2}$ - $o_{5,3}$ - $o_{27,1}$

as Figure 5 depicts by the continuous and respectively discontinuous lines.

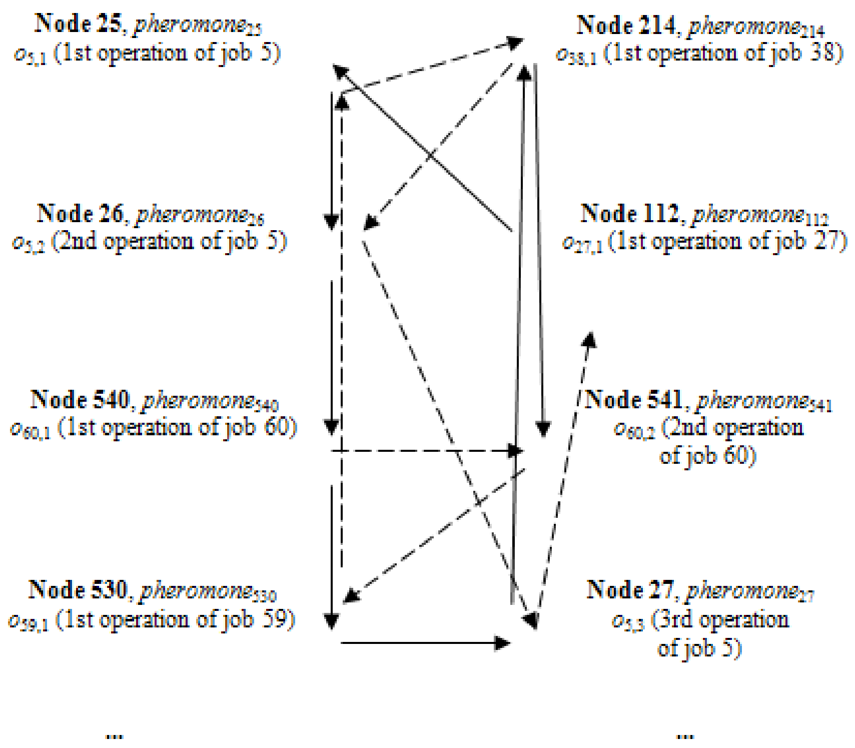
Node 25, *pheromone*$_{25}$
$o_{5,1}$ (1st operation of job 5)
Node 214, *pheromone*$_{214}$
$o_{38,1}$ (1st operation of job 38)
Node 26, *pheromone*$_{26}$
$o_{5,2}$ (2nd operation of job 5)
Node 112, *pheromone*$_{112}$
$o_{27,1}$ (1st operation of job 27)
Node 540, *pheromone*$_{540}$
$o_{60,1}$ (1st operation of job 60)
Node 541, *pheromone*$_{541}$
$o_{60,2}$ (2nd operation of job 60)
Node 530, *pheromone*$_{530}$
$o_{59,1}$ (1st operation of job 59)
Node 27, *pheromone*$_{27}$
$o_{5,3}$ (3rd operation of job 5)
...
...

Fig. 5. Two partial solutions built by two ACO agents for a JSSP instance

A variant of ACO can impose the agents whose current makespan exhibits an a priori maximum value to be rejected from visiting graph and, in that case, the visited nodes will not receive a pheromone update. By this mechanism, the colony is indirectly informed about the poor identified solutions.
The ACO procedure for (MOF)JSSP is the following:

```
1. initialization
    1.1. t ← 1
    1.2. set the parameters ($N_a$ - number of agents, $\alpha$)
    1.3. initialization of pheromone trails
    1.4. initialization of the best current schedule $S_0$
2. while stopping_criterion = false do
       2.1. generate $N_a$ solutions
       2.2. evaluate solutions: $f(S_1), f(S_2), ..., f(S_{Na})$
            $S_b$ ← best solution
       2.3. if $f(S_b) < f(S_0)$ then $S_0$ ← $S_b$
       2.4. update pheromone trails
       2.5. $t$ ← $t$+1
       2.6. if stopping_criterion = true then return the schedule solution $S_0$
```

The advantages of ACO model are manifold:

- the stochastic component allows the agents to build various different solutions and therefore ACO explores a wide search space;
- the agents can be additionally guided to-

ward the promising solutions in the search space by using heuristic information about the problem;

- the agents' experience is used in building solutions in further iterations;
- the collective interaction between agents leads to efficiency and robustness of the solutions.

As a weak point of ACO model we mention the critical aspect of setting adequate values for the parameters for every instance, especially if it is a MOFJSSP one.

On the other hand, ***Particle Swarm Optimization*** (PSO), proposed by Kennedy and Eberhart [29], is an optimization model which simulates the group behavior to efficiently attain a destination in birds flocks, fish schools and other living creatures groups. The agents, named particles, "fly" in the search space following the current optimal particles on basis of an adaptable speed which directs their moves, and on basis of a memory which stores the best visited location in the past by all the agents.

The search efficiency is determined, as in other agent-based models, by the colony of agents, whose speed dynamically updates depending on the system characteristics. The model is similar to GA in the pseudo-random initialization of the population of candidate-solutions and in the searching process based on updating populations in generations. PSO, instead, do not use genetic operators to generate new candidate-solutions.

Particularly for (MOF) JSSP, PSO gradually updates a population of schedules which „moves" in the search space towards an optimal one by modifying the sequences of operations in the schedules.

The main advantages in applying PSO model are the implementation simplicity, the reduced number of parameters to adjust and a wide search space explored. The difficulty, on the other hand, consists in avoiding the local optima.

4 Conclusions

Optimization modeling tools in the literature adequate for the MOFJSSP are numerous and diverse. The conventional ones put the process in the center of the model. Such models are Petri nets, waiting systems, general decision models, logical formulations as STRIPS language, Markov processes, Monte Carlo simulation. Previous studies show that the unconventional methods are generally poor in exploring the search space, which regularly is nonlinear, discrete, contains multiple optima and the global optima are not known in advance. Moreover, for this kind of processes the user often seeks more solutions, and a conventional method generates a single such solution at every execution.

In order to tackle such issues most research was focused on finding other methods, so-called unconventional optimization models. They are procedural models which send in background the process model, the central role in modeling being given to the algorithmic procedure that adjusts the system. These procedural models cover a very active research subject in the last decade for modeling and control of optimization processes. This special kind of approach was imposed by the major difficulty in rigorous mathematical characterization of big complex technological processes behavior, as also MOFJSSP are; in this case, it is not possible an approach which easily covers all the possible states of the system [13]. Such models are evolutionary algorithms and genetic algorithms in particular, the agent-based models (negotiation-based techniques, Ant Colony Optimization, Particle Swarm Optimization and Wasp Behavior Model), the neural networks, the expert systems, the knowledge-based systems and fuzzy techniques.

Based on previous work on optimization models ([14], [15]), the genetic algorithms and agent-based models are the most adequate to handle MOFJSSP, which are quite big and pretty complex processes even for a medium-scale production system.

GA allow finding more diverse solutions in one run, are simple, easy to extend, very suited to difficult optimization processes, hard to analytically handle with.

The multi-agent approach offers another adequate framework for MOFJSSP. It allows using certain complex desirable behaviors emerging when the set of interacting agents is globally considered. Hence, the colony proves to be an intelligent entity when solves a problem, while the individual agents do not have this ability. Most of nature inspired procedural models are good examples of well distributed natural-like multi-agent systems, formed by hundreds or thousands of autonomous agents, robust to loss of individuals and robust to environment changes. The colony acts by coordination of agents' tasks with no direct communication. Additionally, the global performance of the colony proves to be very efficient [25]. The only critical aspect in using agent-based models remains defining efficient rules for interactions, which has a great influence over the model quality.

References

[1] F.G. Filip, "Decision support and control for large-scale complex systems", *Annual Reviews in Control*, vol. 32, no. 1, pp. 61–70, 2008.

[2] F.G. Filip and K. Leiviskä, "Large-Scale Complex Systems", in *Springer Handbook of* Automation, Berlin: Springer Berlin Heidelberg, pp. 619-638, 2009.

[3] P. Borne, D. Popescu, F.G. Filip and D. Stefanoiu, *Optimization in Engineering Sciences,* London: Iste & J.Wiley, 2013.

[4] F.G. Filip, G. Neagu and D.A. Donciulescu, „Job shop scheduling optimization in real-time production control", *Computers in Industry*, vol. 4, no. 4, pp. 395–403, 1983.

[5] M. Kaufmann, *Methods and models of operations research*, vol. II (in Romanian), Bucharest: Ed. Ştiinţifică şi Enciclopedică, 1975.

[6] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, „Optimization and approximation in deterministic sequencing and scheduling: A survey", *Annals of Discrete Mathematics,* vol. 5, pp. 287-326, 1979.

[7] M.L. Pinedo, *Scheduling. Theory, Algorithms, and Systems*, 3rd ed., New York: Springer Science-Business Media, LLC, 2008.

[8] P. Brucker and R. Schlie, „Job-shop scheduling with multi-purpose machines", *Computing* vol. 45, no. 4, pp. 369-375, 1990.

[9] D. Applegate and W. Cook, „A computational study of the job-shop scheduling problem", *ORSA Journal on Computing,* vol. 3, pp. 149-156, 1991.

[10] A.S. Jain and S. Meeran, „A State-of-the-Art Review of Job-Shop Scheduling Techniques", *European Journal of Operations Research*, vol. 113, pp. 390-434, 1999.

[11] M.T. Jensen, *Robust and flexible scheduling with evolutionary computation*, PhD thesis, Denmark, Aarhus University, 2001.

[12] E.S. Nicoară, *Contribuţii privind utilizarea algoritmilor genetici la conducerea ordonanţării flexibile multiobiectiv a producţiei multisortimentale* (in romanian), PhD thesis, Ploieşti: Petroleum-Gas University in Ploieşti, 2011.

[13] I. Dumitrache, *Ingineria reglării automate* (in romanian), Bucharest: Ed. Politehnica Press, 2005.

[14] E.S. Nicoară, „Multi-objective Flexible Job Shop Scheduling Optimization Models (I)", *Economic Insights - Trends and Challenges*, vol. LXIV, no. 2, pp. 79-86, 2012.

[15] E.S. Nicoară, „Multi-objective Flexible Job Shop Scheduling Optimization Models (II)", *Economic Insights - Trends and Challenges*, vol. LXIV, no. 3, pp. 96-104, 2012.

[16] F.G. Filip and B. Bărbat, *Informatică industrială - Noi paradigme şi aplicaţii*, Bucharest: Ed. Tehnică, 1999.

[17] J.H. Holland, „Genetic algorithms", *Scientific American*, vol. 267, no. 1, pp. 44-50, 1992.

[18] L. Davis, „Job shop scheduling with genetic algorithms", in *Proc. the International Conference on Genetic Algorithms and their Applications*, San Mateo, Morgan Kaufmann, 1985, pp. 136-149.

[19] R. Nakano and T. Yamada,

„Conventional genetic algorithms for job-shop problems”, in *Proc. the 4th International Conference on Genetic Algorithms*, San Diego, California, 1991, pp 477-479.

[20] C. Bierwirth, „A generalized permutation approach to job shop scheduling with genetic algorithms”, *OR Spektrum*, vol. 17, pp. 87-92, 1995.

[21] Nicoară, E.S., Filip, F.G., Paraschiv, N., “Simulation-based Optimization Using Genetic Algorithms for Multi-objective Flexible JSSP”, Studies in Informatics and Control, vol. 20, issue 4/2011, ISSN 1220-1766, pp. 333-344.

[22] I. Kacem, „Scheduling flexible job-shops: a worst case analysis and an evolutionary algorithm”, *International Journal of Computational Intelligence and Applications*, vol. 3, no. 4, pp. 437-452, 2003.

[23] H. Fang, P. Ross and D. Corne, „A promising genetic algorithm approach to job shop scheduling, rescheduling, and open-shop scheduling problems”, in *Proc. the 5th International Conference on Genetic Algorithms*, Urbana-Champaign, Illinois, 1993, pp. 375-382.

[24] A. Colorni, M, Dorigo, V. Maniezzo and M. Trubian, „Ant system for job-shop scheduling”, *Belgian Journal of Operations Research, Statistics and Computer Science*, vol. 34, pp. 39-53, 1994.

[25] V.A. Cicirello and S.F. Smith, „Insect societies and manufacturing”, in *the IJCAI-01 Workshop on Artificial Intelligence and Manufacturing: New AI Paradigms for Manufacturing*, The Robotics Institute, Carnegie Mellon University, 2001, pp. 33-38.

[26] M. Dorigo, *Optimization, Learning and Natural Algorithms*, PhD thesis, Politecnico di Milano, Italy, 1992.

[27] G. Theraulaz, S. Goss, J. Gervet and J.L. Deneubourg, „Task differentiation in polistes wasp colonies: A model for self-organizing groups of robot, From Animals to Animats”, in *Proc. the First International Conference on Simulation of Adaptive Behavior*, MIT Press, 1991, pp. 346-355.

[28] V.A. Cicirello and S.F. Smith, „Randomizing dispatch scheduling policies”, in *the 2001 AAAI Fall Symposium: Using Uncertainty Within Computation*, AAAI Press, North Falmouth, Massachusetts, Technical Report FS-01-04, 2001, pp. 30-37.

[29] J. Kennedy and R. Eberhart, „Particle swarm optimization”, in *Proc. the IEEE International Conference on Neural Networks*, Perth, Australia, IEEE Service Center, Piscataway, NJ, 1995, pp. 1942-1948.

Elena Simona NICOARĂ graduated Mathematics-Computer Science in 2000 and Science of Commodities in 1999 at the Petroleum-Gas University (PGU) in Ploieşti, Romania. In 2000 she joined the staff of PGU and currently is teaching assistant in the Computer Science Department. She holds a PhD diploma in Control Engineering since 2011 in the GA-based optimization control field. She authored/coauthored more than 30 scientific papers and over 6 books in artificial intelligence, optimization techniques and evolutionary computation in particular.