Economic Insights – Trends and Challenges	Vol. I (LXIV) No. 3/2012	96 - 104

Multi-Objective Flexible Job Shop Scheduling Optimization Models (II)

Elena Simona Nicoară

Petroleum-Gas University of Ploieşti, Bd. Bucureşti 39, Ploieşti, Romania
e-mail: snicoara77@yahoo.com

Abstract

For the Multi-Objective Flexible Job Shop Scheduling Problems (MOFJSSP), various optimization models have been designed. In the first part of the research (published in no. 2/2012) we presented the multi-sort production systems, the theoretical aspects of MOFJSSP framework, a specific real JSS process in drugs industry based on over 600 tasks to be optimally scheduled (used as a case study for the optimization models) and two conventional optimization models: waiting systems and STRIPS language (an example of logical formulation for scheduling problems). In the current part of the research, other suited models are described, both conventional and non-conventional: Petri nets, general decision models, Markov processes, Monte Carlo simulation and some procedural models (neural networks, expert systems and fuzzy techniques). Other well-known and wide-applied procedural models (namely agent-based models and genetic algorithms) are to be described in the forthcoming part of the research.

Key words: *job shop scheduling, optimization model, time-efficiency, multi-objective*

JEL Classification: *C61; L65*

Introduction

The term Multi-Objective Flexible Job Shop Scheduling Problem (MOFJSSP) covers a particular scheduling problem, which mostly appear in manufacturing area. Here, the jobs to be scheduled are structurally different (regarding the number of operations that compose the jobs and processing order on the machines), there are flexible job routings on the machines and multiple objectives are required.

In the next section many optimization models suited to MOF(JSSP) are described, with the aim to identify the models' limits, advantages and level of appropriateness for specific manufacturing contexts. Some of them are mostly descriptive (graphical, depicting how decisions should be made) and others are mostly normative (analytical, specifying an algorithm to schedule), but no one is exclusively descriptive or normative.

Optimization Models for (MOF)JSSP

The so-called *conventional optimization models* are built on the process model, which is the kernel of the basic concept in these optimization models. The main conventional optimization models for (MOF)JSSP are: the waiting systems, the logical formulations such as STRIPS language (both presented in the first part of the research), the Petri nets, the general decision models, the Markov processes and Monte Carlo simulation.

On the other hand, the *unconventional models* are unconventional technologies which place the process model on a secondary layer; the primary function in modeling is assessed to a procedure that controls the system. This procedure may be easily represented as sequences of instructions, likely to transpose in (exact or heuristic) algorithms. Emergence of these models, named *procedural models*, was imposed by the multiple major difficulties in rigorous mathematical characterization of big complex technological processes (such as MOFJSSP) behavior, where it is not possible an approach which (easily) cover all the possible states of the system (Dumitrache, 2005). These models are preponderantly borrowed from artificial intelligence area, because they handle with big complex problems. Such models are evolutionary algorithms in general and genetic algorithms in particular, agent-based models (negotiation techniques, Ant Colony Optimization, Particle Swarm Optimization and Wasp Behavior Model), neural networks, fuzzy techniques, expert systems and knowledge-based systems.

Petri Nets

A Petri net is a graphical and mathematical state-oriented tool, proposed by Carl Adam Petri (1962) to model, simulate, analyze, design and operate over sequential processes. Owing to ability of a Petri net to assimilate the complex characteristics in discrete-event systems (DES), such as: sequential actions, dependency, concurrency, parallel machines, cycles, conflict, resources sharing and synchronization, this model is pretty adequate for JSS in manufacturing. In Petri nets, the process states, named *places* or *locations*, together with the events, named *transitions*, compose an *oriented bipartite graph* as in figure 1 (where a process with four one-operation jobs is depicted).

A convenient representation for JSS processes uses a place for every resource (machine) and two special places (*start_place* and *end_place*) to specify the "ready job" condition and respectively "finalized job" condition. The tokens inside indicate for the resource places the number of jobs in process on the corresponding machine, and for the mentioned special places the number of ready jobs and finalized jobs. The Petri net in figure 1 presents a capture of a process when three jobs are ready to be processed, one job is processed on the machine correspondent to the place l_1, and no job is finalized. Existence of a token in a resource place indicates that the machine is busy, is not available for processing other jobs.

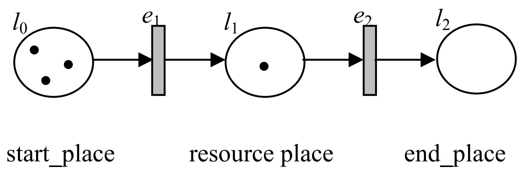
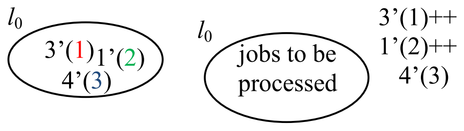


Fig. 1. A Petri net for a production process with 4 one-operation jobs, processed on a machine



a. The compact form **b. The multiset form**

Fig. 2. Representations for start place with colored tokens in a CPN

The system state at a moment in time (or capture of the process), named *marking*, is specified by the number of tokens in every place of the net.

The Petri nets (PN) formalized in 1962 are able to model uni-sort manufacturing processes in a non-timed context - from a logical dynamics perspective. Hence, the times when the events occur are ordered only as a succession (they are not specified) and there are not used time intervals. For the MOFJSS processes, which require a timed context, combing two extensions of the initial PN model, namely *colored PN* and (CPN) and *timed PN* (TPN), is adequate.

CPN, formalized by Kurt Jensen (1979), are high-level Petri nets, where the tokens in places are distinguished by "colors". This model allows natural representations for the multi-sort manufacturing, where the jobs are not identical. In a CPN, every place is labeled with a multiset of elements (Jensen and Kristensen, 2007). For example, the start place for a JSS process where 8 jobs of three different types (1,2,3) are all waiting to be processed, can be represented as in figure 2. The set of colors is $A = \{1,2,3\}$ and the multiset built on A is 3'(1)++1'(2)++ 4'(3).

In the non-timed CPN, the transitions are fired instantaneously. But in the real processes, this is not the case – an operation is processed on a machine for a while, time when the machine is not available for other operations. In the timed context of FJSS systems, one can use the *timed colored PN model* (Jensen, 1992). Here, time is associated to the tokens in places to specify the moments when the jobs are ready for processing and to transitions to specify the time while the machines are unavailable.

A *timed colored Petri net* is defined by a tuple (Jensen, 1991):

TCPN = $(L, E, F, \Sigma, V, C, G, I, \Omega)$,

where each component will be described in the following for MOFJSS processes using the case study in drugs industry presented in the first part of the research.

- L is the set of places. For the case study, $L = \{l_0 =$ "Ready jobs", $l_1 =$ "Supply, weigh", $l_2 =$ "Blend", $l_3 =$ "Granulate", ..., $l_{21} =$ "Finalized jobs"$\}$.
- $E = \{e_1, e_2, \ldots\}$ is the finite non-null set of transitions. A transition is placed between every two consecutive places associated to the machines in every possible route of jobs. For example, a transition e_4 will be placed between l_3 and e_4 to indicate that some jobs follow the machine 4 (Dry up) after machine 3 (Granulate).
- The arcs in $F \subset (L \times E) \cup (E \times L)$ specify flow relations between places and transitions. These arcs translate the technological succession of operations on the machines, specifying all the possible routings of the jobs on the machines.
- The set Σ contains characteristics of the tokens. For our case study, $\Sigma = \{Sort\}$ because product type is the single characteristic of tokens and $Sort = \{1,2,\ldots,16\}$. In correspondence with the specified color in Σ, $V = \{type: Sort\}$. The *type* variable will hence specify the type of product (or sort).
- The coloring function C assigns to places multisets of colors according the relation:

$$C(l) = {}^{++}\sum_{x \in Sort} a(x)'x.$$

- G is the set of guard expressions assigned to transitions, build over the variables. These guard expressions filter the events occurrence and are placed in the Petri net as [*expr*1, *expr*2,…]. For example, the expression [$type \in \{2,8,10,11,12,13,15\}$] associated to transition e_3 forbids transition validation for sorts that are not present in the expression; this is equivalent with the fact that only these types of product can pass over after blending stage to granulating stage. Other types of product pass over to compressing stage instead.
- The marking, which is a system state, assigns to every place a multiset built on Σ. So, a marking is the $|L|$ - dimensional vector:

$$\mathcal{M} = [\mathcal{M}(l_0), \mathcal{M}(l_1), \ldots, \mathcal{M}(l_{21})]^T.$$

I is the initial marking function; for our example,

$$\mathcal{M}_0 = [\mathcal{M}_0(l_0), \mathcal{M}_0(l_1), \ldots, \mathcal{M}_0(l_{21})]^T =$$

$$= [8'(1) ++ 1'(2) ++ \dots ++ 1'(16), \varnothing_\Sigma, \dots, \varnothing_\Sigma]^T.$$

The initial marking for the only non-null place l_0 is depicted in figure 3. The transitions are validated by interpreting the variables in guard expressions. To be valid, a transition must have enough tokens in its previous place and these tokens must have values that satisfy the associated guard expression.

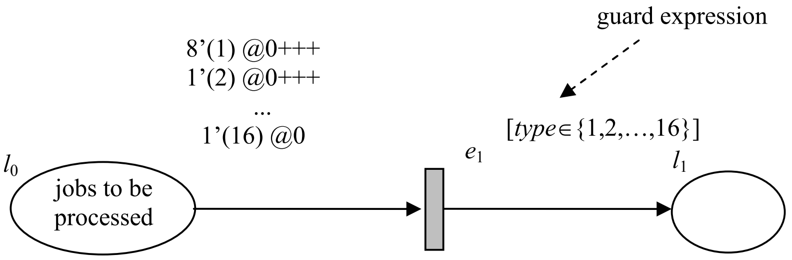


Fig. 3. Section of TCPN for the case study at marking

- Ω, the timing function for the tokens and transitions, is the function that (Jensen and Kristensen, 2007): associates to each token in places a temporal mark to specify the moment when this is ready to be moved (consumed) by a transition firing (as in figure 3), and associates to each transition a duration representing the time interval until production of token at the next place ends (in fact, this is the processing time of the job represented by the consumed token on the machine corresponding to the place after transition, as in figure 4).

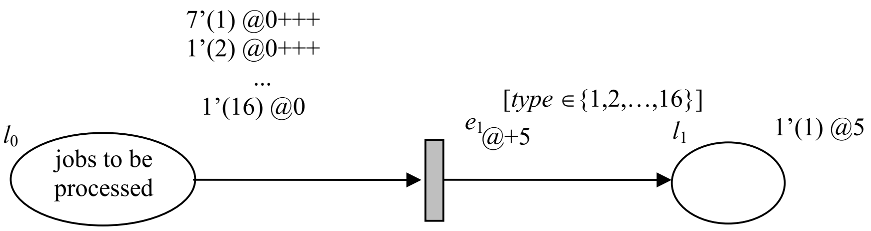


Fig. 4. Section of TCPN for the case study at marking

The TCPN own a global clock, initialized with zero. To model the fact that an operation $o_{i,j}$ last $\tau_{i,j}^{k}$ time units on the resource k, once a transition fires we create temporal marks for the consumed tokens with $\tau_{i,j}^{k}$ greater than the clock value when the transition fired. Supplementary, to the transitions we attach a temporal mark $@+\tau_{i,j}^{k}$, as in figure 4.

When a job processing on a machine ends (in other words the global clock recorded passing the $\tau_{i,j}^{k}$ time units), the machine is unloaded and the job moves to the queue of the next machine in its route. The initial marking $\mathcal{M}_0$ associated to value zero of global clock is that in figure 3. In this context, the jobs associated to the place l_0 are available for transitions to time zero. The new marking, after transition e_1 fired in interpretation <*type* = 1> is that in figure 4. A token of type 1 migrates from l_0 in l_1, and to transition e_1 we associate a delay +5 to indicate that the consumed token will not be available for another transition for 5 time units. This event denotes that supply - weigh operation starts for a series of type 1, the corresponding machine becomes unavailable for 5 minutes and the series will be available for other events at time 5.

The evolution of states in TCPN is controlled by the global clock and is determined by the validation and firing rules of transitions. The system stays at a step of global clock while valid interpretation elements (with necessary adequate colored tokens) exist and they are ready to execute. When no interpretation element can be executed at current step of the clock, the system advances the clock to the next step where interpretation elements can be executed (Păstrăvanu et al., 2002). The final marking of TCPN is:

$$\mathcal{M}_{final} = [\mathcal{M}_{final}(l_0), \mathcal{M}_{final}(l_1), \dots, \mathcal{M}_{final}(l_{21})]^T =$$

$$= [\varnothing_{\Sigma}, \varnothing_{\Sigma}, \dots, \varnothing_{\Sigma}, 8'(1) ++ 1'(2) ++ \dots ++ 1'(16)]^T.$$

A production cycle is ended when $\mathcal{M}_{final}(l_{21}) = \mathcal{M}_0(l_0)$, in other words when the final marking associated to the last place is identical, as multiset, with the initial marking of the first place (all the jobs passed over all the processing stages).

In the TCPN model, schedules are the sequences of fired transitions. In every firing, the type of product and the value of global clock are kept. Therefore, we can state that by modeling the scheduling with TCPN we search a sequence of transition firings which converts the initial marking $\mathcal{M}_0$ in the final marking $\mathcal{M}_{final}$ such that the makespan is minimum.

In order to obtain the optimal schedule one can apply diverse techniques to determine the firing transitions sequence. The classical method uses the accessibility graph. Other methods use decision rules to solve conflicts or heuristic search in the accessibility graph (most of them based on branch and bound algorithms or even genetic algorithms (Chiu and Fu, 1997; Tang and Zhong, 2006)). In the last case, genes encode transitions and chromosomes encode transitions sequences.

Generally, for the small and not so complex scheduling, analytical solutions are provided, and for the big complex problems, as most of real world MOFJSSP are, computer-aided solutions are adequate.

The main advantages of Petri nets express in modeling and analysis of concurrent systems, as we seen before. The limitations, which narrow their applicability as general valid optimization tools for real world problems, rise from the explosion of states once the system complexity grows, and make the simulation big time consumer.

General Decision Systems

Jay Forrester (1973) proposed the concept of „*industrial dynamics*”, for microeconomic and macroeconomic processes, based on dynamic modeling subordinated to quantitative analysis and to qualitative interpretations. The core element in Forrester modeling techniques is an elementary feedback cycle, called *information-decision-action cycle*, as depicted in figure 5.

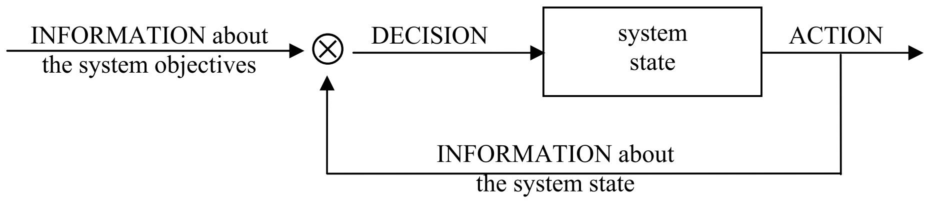

INFORMATION about the system objectives
DECISION
system state
ACTION
INFORMATION about the system state

Fig. 5. The information-decision-action cycle in Forrester dynamic modeling techniques

The global perspective over the system offered by the mutual interconnections existent in the various economic flows, makes the Forrester techniques most adequate to model the general management level in a production organization. The industrial dynamics may be applied also at scheduling level, by specifying supplementary variables and equations.

Another general decision model suited to scheduling is based on *decision trees*. This is a normative model, as it describes "how should decide", rather than "how are made the decisions". The decision trees are composed by decision nodes distributed on hierarchical levels; every descendent of some node represents a decision alternative (at that level) which can be chosen among the others descendents of the same node, depending on the ability to accomplish the settled objective. In figure 6 a partial decision tree associated to the considered case study is presented.

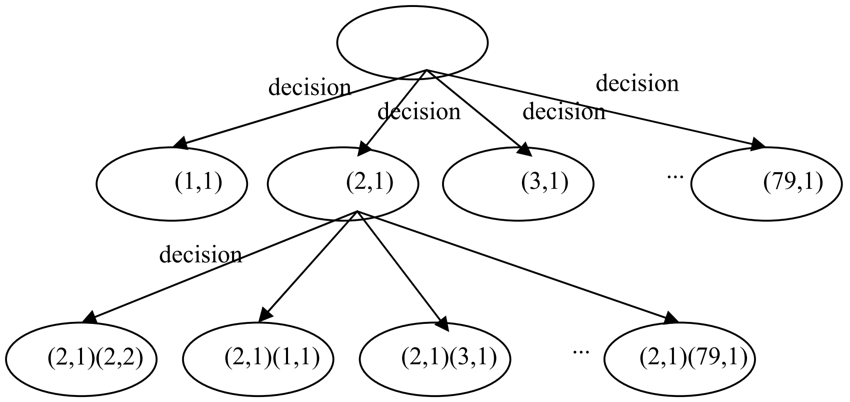


Fig. 6. Three levels from the 606 levels in the decision tree associated to the considered case study

In JSS context, a schedule can be in many distinct states: null schedule, partial schedule (a partial sequence of scheduled operations) or complete schedule (where all the operations in the jobs operations set are included in the processing sequence). The schedule passes from a state to a successor state (descendent state) by making decision to process a certain operation $o_{i,j}$ among the unprocessed operations such that the result schedule remains valid. To every decision a penalty is associated, often determined by the lateness in finalizing the schedule once added the new operation. In the model the objective is to identify an optimal decision sequence (which leads to a minimum global lateness).

This type of model is adequate if the schedule-solution is searched by branch and bound methods, which keep trace of decisions and schedule states for the multiple possible branches, as figure 6 shows. If no intelligent strategy is available to visit the decision tree, an exhaustive tree scan leads to an optimal schedule, but with a high cost. When heuristic (intelligent) methods are used, the cost is highly reduced, because some branches, often many of them, are abandoned in early stages, based on guided search towards promising schedules.

The decision tree model is ordinarily applied for uncertainty contexts (where penalty or profit associated with decisions is probabilistically determined) and is more suitable, from obvious reasons, when the number of alternatives is relative small.

Markov Chains and Monte Carlo Simulation

For the stochastic scheduling one can also use two models become classic: Markov chains and Monte Carlo simulation. *Markov chains* model stochastic processes with discrete states space, where the probability that the system state at a time is a certain one depends only on the system state at the previous moment. Consequently, scheduling modeling with Markov chains allows to consider partial schedule as system state.

Monte Carlo simulation is a wide general model based on searching optimum in a search space using probabilistic sampling by random variables generation, to build new states of the system. Such models for scheduling are simulated annealing and genetic algorithms described in the next part of the research.

Neural Networks, Expert Systems and Knowledge-based Systems, Fuzzy Techniques

Neural networks are a special category of self-learning models, based on training-based learning models used by living organisms. They are complex regression models which learn relationships between inputs and outputs of a system (often nonlinear relationships) if they have enough vast sets of experimental data.

Rabelo (1990) used the first backpropagation neural network for a scheduling process with multiple types of jobs and 3-20 machines, where the network output was a relative order of available priority rules. In JSSP, scientific community claims that the supervised learning neural networks capture the desired input-output relationships by exposing the network to training schemata, but for the most real-world problems the necessary data for an efficient training often are not available (Jain and Meeran, 1998; Jain and Meeran, 1999). Moreover, the tests revealed that a variation of real data against the training data greater than 20-30% often leads to suboptimal results. Another conclusion of the many tests achieved in applying neural networks in scheduling modeling is that they are more efficient combined with priority dispatch rules systems (PDRS).

Expert systems have a limited effect in solving the most real JSS problems, but they can be used to model such processes. ISIS, a well-known expert system for JSSP, uses constraint-driven reasoning. MPECS (Multipass *Expert* Control System), an integrated simulation planner, is a hybrid decision system based on current reaction in manufacturing floor and uses back-chaining and forward-chaining to choose a small set of potential good priority rules and heuristics in the knowledge base. The selected rules are then evaluated by a simulation determined by the expert system (Jensen, 2001).

Fuzzy logic is applied especially to scheduling where uncertain processing times are present. Generally, fuzzy techniques are more efficient when hybridized with constraint relaxation, knowledge-based systems, priority dispatch rule systems or genetic algorithms.

Conclusion

The level of complexity associated to the evolution in time of MOFJSS systems being known, analytical model formulations are not available. But the Petri nets, the procedural models (which include the agent-based models, the evolutionary algorithms, expert systems, fuzzy techniques and neural networks), the waiting systems, the decision models, the logical particular formulations, the Markov processes and the Monte Carlo simulation are all suitable to model these state-event systems.

The standard Petri net model, formalized by Carl Adam Petri in 1962, can handle many characteristics in scheduling processes, such as: sequential actions, dependency, concurrency, parallel machines, cycles, conflict, resources sharing and synchronization, but it is not able to model multi-sort manufacturing processes in a timed context. For MOFJSS processes timed colored Petri nets are used instead. These state-oriented tools are very suited to small-medium instances of MOFJSSP, due to their abundant information about the system evolution and the optimal solution(s). However, an important aspect to consider for the real world MOFJSSPs, is that Petri net models are very expensive in computational time and memory, due to the high explosion of states once the system complexity grows.

The Forrester industrial dynamics model needs for JSSP supplementary variables and equations; it is based on information-decision-action cycles in process flows and therefore is more suitable to model high level management in production organizations. Decision trees, another general decision model, are designed for simple, non-timed and stochastic decision systems. They are properly used for small to medium instances of JSSPs and heuristic methods to visit the decision tree are necessary for an efficient solution.

These general decision models, together with Markov chains, Monte Carlo simulation and fuzzy logic are especially suitable to stochastic scheduling processes and are more limited than other optimization models regarding the process complexity, the scheduling flexibility and the multiple objectives. Neural networks, on the other hand, works better with PDRS and needs vast training data to offer optimal solutions.

As a final conclusion, conventional optimization models (waiting systems, logical formulations, Petri nets, general decision models, Markov processes and Monte Carlo simulation) are more suitable for small and middle-size scheduling processes, and the procedural models are adequate to big complex processes, as MOFJSSPs are.

References

1. Chiu, Y.F., Fu, L.C,.”A GA embedded dynamic search algorithm over a Petri net model for an fms scheduling”, in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1997, pp. 513-518.
2. Dumitrache, I., *Ingineria reglării automate*, Ed. Politehnica Press, Bucharest, 2005.
3. Forrester, J.W. *Industrial dynamics*, Massachusetts Institute of Technology, eight printing, 1973.
4. Jain, A.S., Meeran, S., “Job-shop scheduling using neural networks”, *International Journal of Production Research* 36(5), 1998, pp. 1249-1272.
5. Jain, A.S., Meeran, S., “A State-of-the-Art Review of Job-Shop Scheduling Techniques”, *European Journal of Operations Research* 113, 1999, pp. 390-434.
6. Jensen, K., *Coloured Petri Nets and the Invariant Method*, DAIMI PB-104, Computer Science Department, Aarhus University, Denmark, 1979.
7. Jensen, K.,”Coloured Petri Nets: A High Level Language for System Design and Analysis Advances in Petri Nets”, G. Rozenberg (Ed.): Advances in Petri Nets 1990, *Lecture Notes in Computer Science* 483, Springer-Verlag, 1991, pp. 342-416.
8. Jensen, K., “Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use”, vol. 1: Basic Concepts, *EATCS Monographs in Theoretical Computer Science* 26, Springer-Verlag, 1992.
9. Jensen, K., Kristensen, L.M., *Coloured Petri Nets*, lecture notes, Aarhus University, Denmark, 2007.
10. Jensen, M.T., *Robust and flexible scheduling with evolutionary computation*, doctoral thesis, Dissertation Series DS-01-10, Aarhus University, Denmark, 2001.
11. Păstrăvanu, O., Matcovschi, M., Mahulea, C., *Aplicații ale rețelelor Petri în studierea sistemelor cu evenimente discrete*, Ed. Gh. Asachi, Iaşi, 2002.
12. Petri, C., *Kommunikation mit Automaten*, PhD Thesis, Universitat Bonn, Bonn, West Germany, 1962.
13. Rabelo, L., *Hybrid Artificial Neural Networks and Knowledge-Based Expert Systems Approach to Flexible Manufacturing System Scheduling*, PhD. Dissertation, University of Missouri-Rolla, 1990.
14. Tang, X., Zhong, S., “Timed Colored Petri Nets Based Modelling and Scheduling of Aero-engine maintenance”, *Nature and Science* 4(2), 2006, pp. 46-51.

Modele de optimizare pentru Ordonanțarea Flexibilă Multiobiectiv a producției multisortimentale (II)

Rezumat

Pentru Problemele de Ordonanțare Flexibilă Multiobiectiv a Producției Multisortimentale (POFMOPM) s-au propus diverse modele de optimizare. În prima parte a studiului de cercetare (publicată în nr. 2/2012) s-au prezentat sistemele de producție multisortimentală, aspectele teoretice de bază ale POFMOPM, un proces real specific de acest tip din industria de medicamente bazat pe peste 600 de sarcini de planificat în mod optim (folosit ca studiu de caz pentru modelele de optimizare) şi două modele de optimizare convenționale: sistemele cu aşteptare şi limbajul STRIPS (un exemplu de formulare logică pentru problemele de ordonanțare). În această parte a cercetării sunt descrise şi alte modele adecvate, atât convenționale cât şi neconvenționale: rețelele Petri, modelele generale de decizie, procesele Markov, simularea Monte Carlo şi câteva modele procedurale (rețelele neuronale, sistemele expert şi tehnicile fuzzy). Alte modele procedurale, binecunoscute şi larg aplicate (şi anume modelele bazate pe agenți şi algoritmii genetici), vor fi descrise în următoarea parte, finală, a studiului.