Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Job Batching and Scheduling for Parallel Non-Identical Machines via MILP and Petri Nets

Ralf Fröhlich
Institute of Automation
Hamburg University of Technology
Hamburg, Germany
ralf.froehlich@tu-harburg.de

Matthias Hüsig
Institute of Automation
Hamburg University of Technology
Hamburg, Germany
matthias.huesig@tu-harburg.de

***Abstract*— This paper presents two modeling approaches for batching and scheduling jobs on parallel non-identical machines. For scheduling of jobs the due dates as a hard restriction and machine dependent processing times are considered. Between each of the jobs a setup may be necessary or not, depending on the predecessor. The setup of the machines is done by a single worker who needs to be scheduled as well, since only he can serve one machine at a time. The objective is to maximize the overall workload of the machines meeting the constraint of the due dates. While the first approach uses Mixed Integer Linear Programming to calculate an optimal schedule, the second one uses a simulation, based on colored Petri nets. Both methods are tested and evaluated using illustrative simulation examples.**

***Keywords*— Family Setup Times, Scheduling on Parallel Non-Identical Machines, Mixed Integer Linear Programming, Colored Petri Nets**

I. PROBLEM STATEMENT

We consider the following application of selecting, batching and scheduling. Suppose we select a number of jobs out of a stock with N_{st} jobs. These jobs are processed by a set of N_m machines which are used to perform the same kind of processing on a number of jobs without preemption. Due dates are assigned to the jobs and between each of the jobs a fixed setup time may be necessary or not, depending on the predecessor. The setup of the machines is done by a single worker. He can be seen as an exclusive resource and can only set up one machine at the same time. In addition the machines are non-identical and therefore the processing time of the same job on different machines may differ as well. Therefore the problem turns up to be a problem of scheduling and batching N jobs on N_m non-identical parallel machines with fixed family setup times. The N_{st} jobs are stored in a common stock and need to be allocated to one of the machines. At the same time the jobs on each machine need to be sequenced. The worker needs to be scheduled to perform the setup of the machines if necessary. The objective is to schedule the jobs and the worker to maximize the work load of the machines under the hard restriction of the due dates. In this paper two approaches are presented to perform the scheduling using a Mixed Linear Integer Programming model resp. a Petri net simulation. For the jobs, the machines and the worker the following characteristics can be stated:

Job characteristics:
C1. No pre-emption (i.e. interruption) of jobs is allowed.
C2. Jobs may be started at any time.
C3. A due date is associated to each job.
C4. Processing time of a job is machine dependent.

Machine characteristics:
C5. A machine may process only one job at the same time.
C6. Machines are available at all times.
C7. Machines may be idle within the scheduled period.
C8. Machines have setup times.

Worker characteristics:
C9. The worker can only serve one machine at a time.
C10. Each setup must be processed to completion.

Both of the presented approaches need to perform the following tasks while finding a feasible and suitable schedule:

- Select the jobs out of the stock to be processed in a production period (e.g. a day).
- Select the machine out of the set of machines on which the job has to be processed.
- Determine the order of the jobs on each machine according to the setup times.
- Schedule the worker to perform the necessary setup.

The schedule needs to be calculated for each production period. At the end of this paper the presented approaches will be validated and compared to each other by using real life data in different machine settings.

II. STATE OF THE ART

Scheduling of jobs on parallel machines and scheduling with fixed family setup times were studied during the past decades. A good overview can be found in [1], [2] and [5].

However the given problem, especially the concurrence of family setup times for the jobs and machine dependent processing times makes it difficult to use an established batching and scheduling procedure. Since two approaches for the given problem are presented in this paper, the following sections give an overview of the current work on related problems.

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

A. MILP Batching and Scheduling

The given problem of scheduling on non-identical parallel machines is a combinatorial complex problem which is NP-hard, even without the considered setup operations [5]. It is therefore difficult to solve or even incomputable in general. To deal with the complexity it is necessary to decompose the overall problem into sub problems to reduce the complexity. First it is useful to build batches which include only jobs that are members of the same setup family [1], since this reduces the necessary setup operations per production period. Jobs with a due date within the production period need to be processed in any case. Selecting the jobs out of the stock and form the batches can be seen as a kind of knapsack problem, while some of the jobs need to be performed and others might be selected or not according to some optimization function. Knapsack optimization is a well studied problem; see e.g. [5] or [6].

The second sub problem is the allocation of the formed batches to the machines considering the exclusive resource of the worker who performs the setup of the machines. Since the processing time of the jobs on the machines may be different, the processing times of the batches on the machines are different as well. The scheduling of the batches and the allocation can be seen as a scheduling problem of parallel unrelated machines. Each of two sub problems is solved for a complete production period. However since the optimization is done for the two sub problems separately, the global optimum for the overall problem of scheduling is not guaranteed, since it might be advantageous to split batches or to perform the optimization for the whole stock and not only for the considered production period.

B. Petri Net based Approaches

In production planning Petri nets are used to model and analyze discrete event systems like production systems [13]. Petri nets provide a very general framework to model timed discrete event systems with parallel processes and mutual exclusive resources. One typical modeling method, the so called “A-Path” modeling, to create Petri nets for discrete event control is described in [13] and is adjusted to colored Petri nets among others by [8] getting compact Petri net models.

Controllers can be implemented as Petri nets as described in [9]. All this is used to analyze discrete event systems by simulation. As described in [12] there are also some approaches which use the reachability graph in order to analyze discrete event systems. This graph can be used to create an optimal controller by supervisory control. In [10] the generation of schedules by means of the reachability graph analysis is compared with a method which generates schedules by simulation. The computational effort to generate a supervisory controller out of the reachability graph for event driven systems increases very much with the size of the problem. Therefore it is not suitable to solve the considered problem in adequate time. Petri net simulations are suited to evaluate the performance of heuristic scheduling, priority and dispatching rules [11], [7]. In [14] the main idea of using the request allocation principle for the controller design is presented. The developed Petri net approach is based on this principle.

III. Batching and Scheduling with MILP

The problem of selecting, batching and scheduling jobs for a given production period T_p can be separated into two sub problems. First the selection of the jobs out of the stock and the composing of the batches are considered. These batches shall contain only jobs which require the same setup on the machines. Second the task of allocation and sequencing of the batches on the machines is done by an optimization.

A. Batching via MILP

Out of the stock of N_{st} jobs the N_{pp} jobs, which need to be processed in any case in the production period, since their due date is within the production period, are used to compose N_{ba} batches, with $N_{ba} \leq N_{pp}$. The remaining N_{re} jobs with a due date later than the considered production period are used to top up the built batches if the required setup fits the setup of one of the N_{ba} batches. The processing time of all batches needs to match the production period. The decision which of the remaining jobs is used to top up the batches can therefore be stated as a Knapsack optimization problem. The aim should be to allocate as many of the remaining jobs to the composed batches as possible. The problem can be formulated as a selection maximization:

$$\max\left\{\sum_{i=N_{pp}+1}^{N_{pp}+N_{re}} s_i\right\} \tag{1}$$

there the Boolean s_i stands for the allocation of the i^{th} job to a matching batch. The capacity constraint that the processing time of all N_m machines should be greater than the overall batch processing time can then be stated as:

$$\sum_{i=1}^{N_{pp}}\sum_{m=1}^{N_m} p_{im} + \sum_{i=N_{pp}+1}^{N_{pp}+N_{re}} s_i \sum_{m=1}^{N_m} p_{im} \leq N_m \cdot \left(N_m \cdot T_p - N_{ba} \cdot T_s\right) \tag{2}$$

there p_{im} is the processing time of job i on machine m. This assures that at the end of the production period the batches are processed completely. Note that the setup time of the machines T_s need to be considered only once per batch. Since the processing time for the jobs may be different on different machines, equation (2) estimates the processing time of the batches at all N_m machines.

B. Scheduling via MILP

The formed batches are used for the scheduling. The processing time of a batch j on machine m is stored in p_{jm}, while T_s stands for the setup time for each batch. The optimization goal of minimizing the end of the last processing time can then be stated as:

$$\min_j \max\left\{ t_j + p_{jm} z_j^m \right\} \quad . \tag{3}$$

For the machine batch allocation

$$z_j^m = \begin{cases} 1 & \text{if batch j is allocated to machine m} \\ 0 & \text{otherwise} \end{cases}$$

is used and t_j marks the starting time of a batch j. For all batches the setup of the machine needs to be done in front of the processing, therefore:

$$t_j - T_s \geq 0. \qquad (4)$$

Since each batch needs to be allocated to one machine only:

$$\sum_{m=1}^{N_m} z_j^m = 1 \qquad \forall \quad j \in 1, \dots, N_{ba} \qquad (5)$$

needs to be fulfilled. Introducing a sequence variable y_{jk}, for the batches with

$$y_{jk} = \begin{cases} 1 & \text{if batch j starts before batch k} \\ 0 & \text{otherwise.} \end{cases}$$

the sequence condition that

$$y_{jk} + y_{kj} = 1 \qquad \forall \quad j,k \in 1, \dots, N_{ba} \qquad (6)$$

can be introduced, which assures that a batch starts either before or after another batch. For the starting times four conditions need to be fulfilled. If the batches are allocated to the same machine, then the conditions

$$t_j - (t_k - T_s) \leq M \cdot (1 - y_{jk}) \qquad (7)$$
$$t_k - (t_j - T_s) \leq M \cdot (1 - y_{kj}) \quad \forall \quad j,k \in 1, \dots, N_{ba}$$

assure that between two starting times of the batches the later one does not start before the setup is made. M is a sufficiently large number compared to the starting times. This models the exclusive use of the setup resource, see Fig. 1a.

In addition, if the two batches are processed on the same machine the processing time of the earlier one needs to be considered:

$$t_j - (t_k - p_{jm} - T_s) \leq M \cdot (3 - z_j^m - z_k^m - y_{jk}) \qquad (8)$$
$$t_k - (t_j - p_{km} - T_s) \leq M \cdot (3 - z_k^m - z_j^m - y_{kj})$$
$$\forall \quad j,k \in 1, \dots, N_{ba}$$
$$m \in 1, \dots, N_m$$

The timing situation for this is depicted in Fig. 1.b.

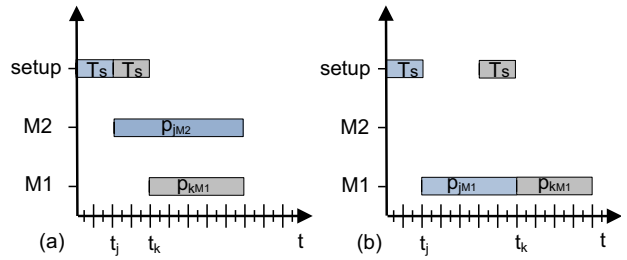

Figure 1. Timing constraints for distinct (a) and identical (b) machines

IV. Batching and Scheduling with Petri Nets

A schedule for the given problem is generated by using a Petri net simulation which models the controller and the plant. The plant is modeled as a discrete event system and for the controller dispatching rules are used. The plant is implemented as a modular scenery in which the number of machines can be easily adapted. The simulation environment is implemented

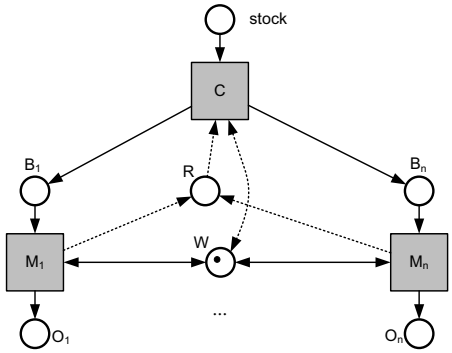

Figure 2. Main level of the hierarchical Petri net model with a controller C, machines M and places in between. Input buffer B, output buffer O, request place R and the worker W.

using colored and timed Petri nets. This allows to manipulate attributes of tokens and to execute code fragments during the firing of the transitions. In this simulation environment the machines, the worker and the controller are modeled with the A-path method [14]. The simulation is initialized using the jobs to be processed within the production period as these jobs are batched and scheduled on the given machines first. If there is some processing capacity left after the first simulation run the load will be increased repeatedly until the production period is fully utilized.

The process of this Petri net simulation is as follows: Each machine generates requests for new jobs if idle. The controller allocates jobs, forms batches and passes them to the machines. The coactions of requests, allocations and the simulation of the plant generate the machine schedule during the simulation run. If the simulation reaches the termination condition, i.e. the production capacity is fully used up, a valid schedule is found for the scheduling problem. The simulation model consists of two parts. One part is a model of the production plant that is used to keep the boundary conditions. The other part contains the controller of the production plant. Fig. 2 shows the main level of the simulation model. It shows circles named places, directed edges named arcs and grey boxes named modules. Places and arcs are normal Petri net structures. The modules are structural Petri net elements and contain subnets. There are three different types of places between the controller C and the machines M in the Petri net. Places labeled with B_i represent the buffer of the particular machine i. For every job to be produced on this machine one token is generated in this place. If a machine runs idle a request is sent to the controller in order to get a new batch. For every request one token is generated in the place labeled R. These requests are produced by the machines which act as event generators. The controller handles this request directly and determines heuristically the best batch for this particular machine to be processed next. After the evaluation of the batches new jobs are released to the machine if the worker is available. The availability of the worker is tested by connecting the place W and the controller C with a test arc. For a better understanding of the operating principle of the controller, the subnet of the controller module C is depicted in Fig. 3. It shows the part of the controller which processes the requests of machine 1. Some of the Petri net elements are drawn in grey others in black.

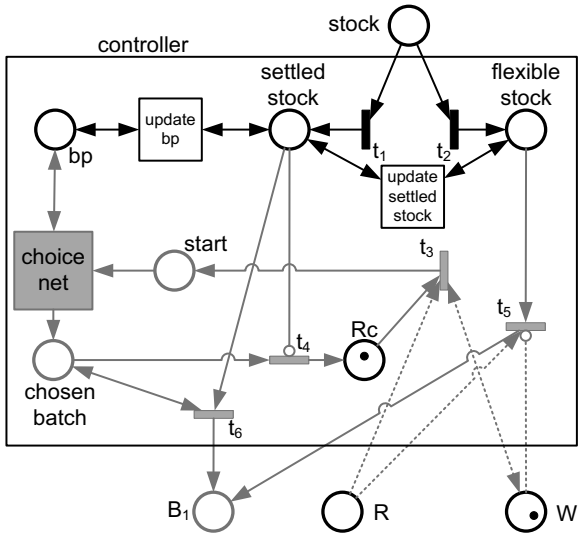

Figure 3. Subnet of the controller module C. Input jobs in stock and requests R, outputs machine buffer B.

The black elements are used for all request replies, the gray ones just for the decision of machine 1. At the beginning of the simulation the data of all jobs in the stock is read in. Jobs are classified into jobs with a due date in the current production period and a due date in future. Depending on the due date of the job transition either *t1* or *t2* is activated, respectively. Each job is represented by a token. The first group of jobs is stored in the place named *settled stock.* Jobs in this place are used as the basis for the scheduling algorithm and have to be processed within the production period. Jobs in the place *flexible stock* can be used to fill idle times and to enlarge the scope of work in the settled stock. The jobs within the settled stock are used to compose batches which require the same setup. At the beginning of the simulation run the jobs can be transferred from the flexible to the settled stock. This is done in the module *update settled stock*. All jobs in the place *settled stock* belong to one of the batches. For each batch the properties are determined like the numbers of jobs or the average processing time. For each batch a token with the batch properties is generated during the read in of the jobs and stored in the place *bp*. Later on, the stored information is used to allocate the jobs and to make sure that each setup is used only once within a production period.

Requests for new jobs are generated by the machines and handled by the controller. The worker has to retool the machine first if a new batch should be processed. The transition *t3* is only enabled if there is a token at the place *W*, at the place *Rc* and the place *R* otherwise transition *t5* is enabled. If *t3* fires, the *choice net* is activated and a new batch is determined. The *choice net* evaluates the remaining batches. For each machine a choice net has to be implemented and be parameterized according to the machine properties. The decision which batch to be processed next depends on the machine type and is done via a priority dispatching rule. After choosing the best batch a token is generated in place *chosen batch* containing the number of the batch to be processed next. If the value of the attributes in *settled stock* and *chosen batch* matches, transitions *t4* or *t6*

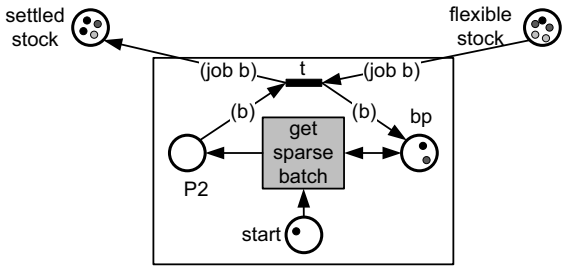

Figure 4. Module update settled stock enlarge sparse batches

are activated or deactivated, respectively. Transition *t6* fires until all jobs related to this batch are released. After this transition *t4* is enabled, the token in place *chosen batch* is removed and added to place *Rc*. This place denotes that the controller is available for the next request. Transition *t5* fires if the worker is not available, i.e. there is no token in the place *W* at the time of a request. The controller releases then a new job from the *flexible stock* if the setup of the machine and the one required by the job match. This bridges the time span until the worker is available again. The request generation is repeated until all jobs in the place *settled stock* are processed. After each simulation run the overall processing time of the selected jobs and the production period are compared. If the processing time is less than the production period, the scope of work is increased. The implemented controller enhances the batch sizes to increase the processing time. Batches with fewer jobs are stocked up first to justify the expense of retooling. Jobs which enlarge the scope of work are reclassified from the *flexible stock* to the *settled stock,* as depicted in Fig. 4. The number of tokens within the place *start* defines the number of additional jobs to be processed. The module *get sparse batch* contains a sort of algorithm which returns the batch with the smallest number of jobs. After that the transition *t* can fire if the attribute *b* of one of the tokens in place *flexible stock* is similar to the attribute b of the token in place *P2*. The job is then added to the place *settled stock*. After this modification the properties of the setup family have to be updated. Afterwards the next sparse batch is determined and enlarged.

The worker represented as a token in place *W* is modeled as a parallel mutual exclusion (PME) in the Petri net. Two processes designed as A-paths both need the resource worker W exclusively (Fig. 5). If a setup is started in one of the machines the token in place W is removed. In this case the transition for starting the setup of the other machine is not able to fire. After the completion of setup the stop transition fires and a token is generated in place *W*.

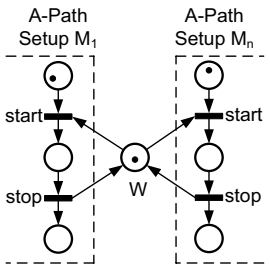

Figure 5. Worker modeled as parallel mutual exclusion (PME)

V. Evaluation of Results

For comparing the two presented approaches three different test scenarios are calculated. The goal is to obtain schedules maximizing the overall workload of the machines maintaining the due dates of the jobs. For this batches need to be built, allocated and to be sequenced on the machines. While the MILP optimization handles the data for the whole production period at once, the Petri net simulation runs until the end of the production period is reached. While the batching in the MILP method is done prior to the actual scheduling, the Petri net approach batches all jobs dynamically. The static batching of the MILP approach does not take the impact of the imbalance of the processing times on different machine into account. Since the approximated overall batch processing time of equation (2) is only a rough guess, the machine production time may not be used as well as possible. On the other hand the dynamical batching of the Petri net approach makes it necessary to simulate the whole production period repeatedly to get a packed schedule. The worker is considered in the MILP model explicitly while in the Petri net approach the worker is considered only if a machine runs idle. The exclusivity constraint may sometimes only be assured by adding waiting times. The Petri net approach always calculates a feasible schedule in the first simulation which includes at least the compulsory jobs. In the worst case the idle time of the machines due to the worker's occupation is half of the setup time for the two machine case. On the other hand the MILP approach might not be able to find a feasible schedule which ends within the required production period, since it builds the batches prior to the actual scheduling. For all sample data a fixed production period of eight hours is considered. The results overview can be studied in Table I.

The first test scenario is based on real production data from a two machine batching and sequencing problem, arising in a printing workstation in the production of semiconductors. There are 220 jobs in the stock, 93 of them compulsory. The jobs usually need between 2 to 10 minutes to be processed dependent on which of the two machines the processing takes place. Since machine M2 is a more sophisticated one, the processing time on M2 is always equal or smaller than on M1. The setup time between the batches shall be 10 minutes. The resulting Gantt charts for the batched jobs can be studied in Fig. 6. Both approaches achieve the objective that all jobs with a due date within the production period are processed and generate therefore valid schedules. In this example a few characteristics of the two methods can be studied, e.g. that the MILP approach does not use the whole machine processing time. This results in an idle time at the end of the production

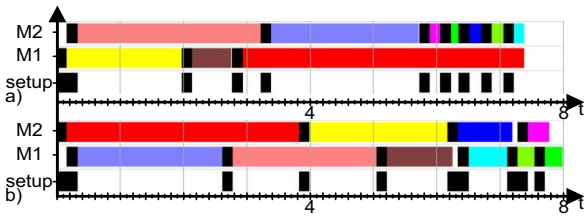

Figure 6. Real test data with 10 batches for MILP a) and Petri net b)

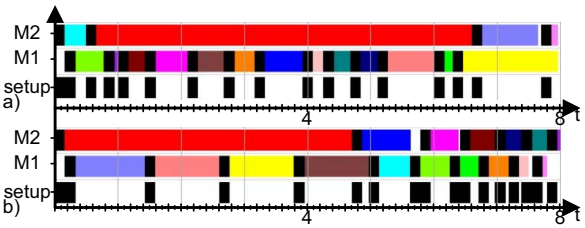

Figure 7. Generated test data with 17 batches for MILP a) and Petri net b)

period due to the bad prediction of the necessary processing time. The Petri net schedule does not use the whole machine time either, since the worker is busy retooling another machine while the end of a batch is reached. This controller can only use this idle time if there are jobs left in the stock which need the same setup. Due to the local decision of the Petri net approach, made when a machine runs idle, the allocation of the batches and the machines is not done in an optimal way. Since the optimization of equation (3) is doing the allocation for the whole production period at the same time, the machines are used for the most suited batches.

The results for the second test scenario are depicted in Fig. 7. In this case 17 different batches need to be processed during the production period, while 180 jobs are considered. The data for this example is generated using an unequally distributed number of jobs per setup family. Both approaches generate a feasible schedule. On the one hand the Gantt chart generated by the Petri net approach shows the result of preferring the big batches at the beginning of the production period. The filling up procedure becomes much more complicated for small batches at the end of the production period, since there are not enough jobs remaining in the stock. On the other hand the MILP procedure does not even consider the size of the batches for the selection from the stock, but prefers small jobs in the optimization equation (1). Due to that the remaining jobs in the stock may be less but need more processing time than the ones left by the Petri net controller. In Fig. 8 the results for the third sample data set are shown. Here the doubled real world data were used to calculate a schedule for a three machine case, while M2 has the same characteristic as M3. As the number of machines and jobs increases the complexity increases as well. However for the presented data the schedule was still computable in reasonable time, this is mainly because the compulsory jobs still have a big share and therefore keep the knapsack problem for the MILP approach small. Also, the

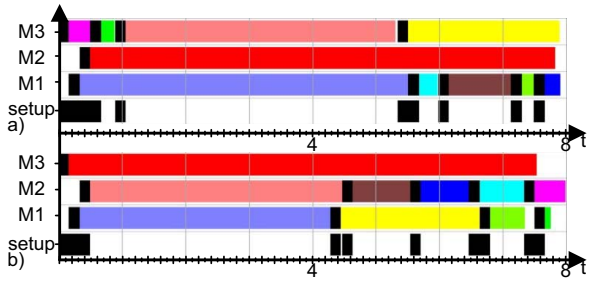

Figure 8. Test data for 3 machines with 10 batches for MILP a) and Petri net b)

TABLE I. TEST SCENARIOS AND RESULTS FOR MILP AND PETRI NET APPROACH

Approach	Scenario	Jobs processed/left	Work-Load	Jobs processing time M1+M2(+M3)	Jobs processing time Left M1/M2/M3
MILP	220 jobs (93 compulsory)	148/72	91.03%	14.56h	11.09h/10.72h
Petri net	10 setups 2 machines	143/77	96.41%	15.43h	8.98/8.13h
MILP	180 jobs (86 compulsory)	135/45	98.18%	15.71h	7.09h/6.89h
Petri net	17 setups 2 machines	125/55	96.26%	15.4h	6.4h/5.79h
MILP	440 jobs (186 compulsory)	210/230	96.21%	23.09h	27.22h/24.75h/24.75h
Petri net	10 setups 3 machines	202/238	94.31%	22.63h	26.53h/23.72h/23.72h

number of batches to be scheduled is constant and there are still only two different kinds of machines since M2 and M3 are identical. The computational time for the Petri net schedule increases as well but its amount is still reasonable. The MILP approach only needs a few seconds for the calculation of the three schedules shown in Fig. 6 to Fig. 8. For the Petri net approach the computational effort increases mainly due to the "up stocking" loop, which is performed repeatedly until the final schedule is found. However the first feasible schedule is found in less than 10 sec. for all of the considered data. As can be studied in Table I, both approaches calculate nearly fully stretched workloads for the machines.

VI. CONCLUSION

This paper presents two scheduling approaches for the job batching and scheduling problem with family setup times on parallel unrelated machines. In addition the setup of the machines is done by a single worker and needs to be considered as well. The presented approaches use a mixed integer linear programming and a Petri net model to solve the scheduling problem. Handling the task quite differently, both approaches calculate valid schedules for a given production period. The performance of the two methods are validated and compared using a real world problem. The objective of maximizing the workload is considered by both approaches by increasing the processed jobs. The MILP method maximizes the number of the processed job despite the job processing time. The Petri net approach increases the number of processed jobs by increasing the batch size of the smallest one.

The determining of the batch size is done prior to the scheduling and sequencing in the MILP method using an estimated workload. This leads to the situation that the machine time may not be used efficiently or the batches may not be processed completely. Here a sequence of repeated optimizations, updating the estimated workload, would help but it increases the computational expenses a lot.

On the other hand the Petri net approach batches all jobs dynamically and determines a feasible schedule within the first simulation run. This first schedule includes the compulsory jobs and is then used to add proper jobs successively. However, this is done in a pragmatic fashion which does not guarantee the efficient use of the worker.

For the implementation of both approaches standard software tools and PC's were used. There can be made further investigation about the worker schedule since this is not considered in both approaches at the moment. Imaginable would be an equally distributed workload or the incorporation of breaks for the worker.

For the given problem both approaches lead to feasible schedules. Considering an increasing complexity for the given problem, e.g. if the number of jobs or machines increases, the Petri net approach is to be preferred. This is due to the fact that a feasible schedule is calculated within the first simulation run. The MILP approach however can be used to check the performance of the Petri net approach and to enhance the used request allocation principle of the Petri net approach.

REFERENCES

[1] T. C. Edwin Cheng, Jatinder N. D. Gupta, Guoqing Wang, "A review of flowshop scheduling research with setup times", Production and Operations Management, Vol. 9, No. 3, 2000, pp. 262-282

[2] C. T Ng , Mikhail Y. Kovalyov, "Batching and scheduling in a multi-machine flow shop", Journal of Scheduling, Vol. 10, No. 6, December 2007, pp.353-364

[3] Shisheng Li, Jinjiang Yuan, "Parallel-machine parallel-batching scheduling with family jobs and release dates to minimize makespan", Journal of Combinatorial Optimization, Springer online, May 2008

[4] Order Scheduling Models An Overview, Multidisciplinary Scheduling: Theory and Applications. 1st International Conference, MISTA '03 Nottingham, UK, 13–15 August 2003

[5] J. Y.-T. Leung, "Handbook of Scheduling", Chapman & Hall/CRC, 2004

[6] "ILOG OPL Development Studio 5.2 - Language User's Manual ", ILOG, 2007, www.ilog.com

[7] Mejia Delgadillo G., Poensgen Liano S, "Scheduling Application Using Petri Nets", 19th International Conference on Production Research, August 2007

[8] M. Aguiar, R. Barreto, R. Caldas, J. Edgar, and C. Filho, "Modeling and analysis of Flexible Manufacture Systems through hierarchical and Colored Petri Nets", Industrial Technology, 2008. ICIT 2008. IEEE International Conference on, pp. 1–6.

[9] M. Dahms, and M. Schmidt, "Modeling of dispatching-rules for job shop scheduling in manufacturing systems - a Petri net approach", Systems, Man and Cybernetics, 2005 IEEE International Conference on, vol. 3, pp. 2025-2030 Vol. 3.

[10] Hehua Zhang, Ming Gu, and Xiaoyu Song, "Modeling and Analysis of Real-Life Job Shop Scheduling Problems by Petri nets", Simulation Symposium, 2008. ANSS 2008. 41st Annual, pp. 279–285.

[11] Chun Wang, H. Ghenniwa, and Weiming Shen, "Heuristic scheduling algorithm for flexible manufacturing systems with partially overlapping machine capabilities", Mechatronics and Automation, 2005 IEEE International Conference, vol. 3, pp. 1139-1144 Vol. 3.

[12] W.M. Wonham, Supervisory Control of Discrete-Event Systems: Unversity of Toronto, 2008.

[13] M. Zhou, and F. DiCesare, Petri net synthesis for discrete event control of manufacturing systems. Boston: Kluwer Acad., 1993.

[14] W. Meyer, and C. Fiedler, "Mission control by coordinating shared resources", System of Systems Engineering, 2006 IEEE/SMC International Conference on, 2006, pp. 13 pp.-.