MISTA 2009

Ant Colony Optimization Approaches for Scheduling Jobs with Incompatible Families on Parallel Batch Machines

Lars Mönch • Christian Almeder

Abstract In this paper, we suggest an Ant Colony System (ACS) to solve a scheduling problem for jobs with incompatible families on parallel batch machines. We are interested in minimizing total weighted tardiness (TWT) of the jobs. Problems of this type have practical importance in semiconductor manufacturing. The ACS scheme includes an efficient local search technique based on swapping jobs across different batches of the same family. A comparison of the suggested ACS scheme with a dispatching rule based approach and a genetic algorithm (GA) from previous research is performed based on stochastically generated test instances. It turns out that the ACS approach slightly outperforms the GA with respect to solution quality but it provides the results using considerably less computational time. We describe extensive computational experimentation to find appropriate parameters for the ACS scheme. Finally, we also present a MAX-MIN Ant System (MMAS) and compare its performance with the ACS approach. MMAS provides basically the same solution quality, but it requires much more computational efforts.

1 Introduction

This research is motivated by a scheduling problem found in the area of diffusion in a semiconductor wafer fabrication facility (wafer fab). In the diffusion area, there are several diffusion furnaces which can be operated in parallel and can process several jobs simultaneously. However, due to the different chemical nature of the processes, only jobs of the same job family can be processed together. Carefully choosing which products to batch together and which order these jobs should be sequenced is of high importance as the processing times in the diffusion furnaces can take up to 10 hours, versus a 1 hour processing time in other wafer fab processes. Therefore, a proper scheduling of the machines dedicated to the diffusion process steps has a great impact on the performance of the entire wafer fab. Given that many wafer fabs are driven by on-time delivery, the performance measure of interest is TWT. This performance measure is given as $\sum w_{js}T_{js}$, where we denote by w_{js} the weight or priority for job j of family s. The tardiness of job j of family s is given by $T_{js} := max(C_{js} - d_{js}, 0)$, where we denote by C_{js} the completion time for job j of family s in the schedule and d_{js} represents the due date for job j of family s.

In this paper, we extend previous research on parallel batch process machines using genetic algorithms (GA) [1]. We suggest an ACS and a MMAS scheme. In contrast to other ACO approaches for parallel machines we use a pure sequencing representation that includes the assignment of the jobs to batches and machines only indirectly. Our ACO schemes are hybridized with a local search technique that changes the content of the batches across

Lars Mönch
University of Hagen
E-mail: Lars.Moench@fernuni-hagen.de

Christian Almeder
University of Vienna
E-mail: Christian.Almeder@univie.ac.at

machines and also the sequence of batches on each machine. We show that ACS slightly outperforms the GA approach with respect to solution quality. But ACS requires considerably less computing time. The performance of MMAS is similar to ACS but much more computing time is required. This kind of comparison between different ACO schemes for the same scheduling problem is rarely discussed in the literature.

The paper is organized as follows. In the next section, we describe the problem under consideration in detail, present a Mixed Integer Programming (MIP) formulation, and discuss related literature. Then we describe our ACS scheme and the MMAS approach. We present results of computational experiments with both heuristics in the last section of this paper.

2 Problem Description and Related Literature

2.1 Problem Setting

For the problem being researched in this paper the $\alpha \mid \beta \mid \gamma$ notation is:

$$Pm \mid p\text{-}batch,\ incompatible \mid \sum w_j T_j \ . \tag{1}$$

There are *f* families of jobs and *m* parallel machines. Family s has n_s jobs. The total number of jobs is *n*. All jobs of family *s* have the same processing time. The jobs that are waiting to be processed must be batched by family, depending on the maximum batch size *B*. Each single job has a weight w_{js} and a due date d_{js}. Our objective is to minimize the TWT measure. Scheduling problem (1) is NP hard because the scheduling problem $1 \mid\mid TWT$ is NP hard. Therefore, we have to look for efficient heuristics.

We present a MIP formulation for problem (1) in order to assess the solution quality of the suggested heuristics for small size test instances. We start with introducing the necessary index sets:

$b = 1,\ldots,n_b$	batch index,	$j = 1,\ldots,n$	job index,
$s = 1,\ldots,f$	job family index,	$l = 1,\ldots,m$	machine index.

Furthermore, we present the necessary parameters:

B	maximum batch size,	p_s	processing time of family s,
d_j	due date of job j,	w_j	weight of job j,
e_{js}	$\begin{cases} 1 & \text{if job } j \text{ belongs to family } s \\ 0 & \text{otherwise} \end{cases}$,	M	large number.

We consider four decision variables. The completion time of the *b-th* batch on machine *l* is denoted by C_{bl}. The tardiness of job *j* is denoted by T_j. C_{bl} and T_j are real-valued variables. Furthermore, the following two binary variables are used:

$$X_{jbl} := \begin{cases} 1 & if\ job\ j\ is\ assigned\ to\ the\ b\text{-}th\ batch\ of\ machine\ l \\ 0 & otherwise \end{cases},$$

$$Y_{bls} := \begin{cases} 1 & if\ batch\ b\ of\ machine\ l\ contains\ only\ jobs\ of\ family\ s \\ 0 & otherwise \end{cases}.$$

The present scheduling problem is formulated as follows:

$$\min \sum_{j=1}^{n} w_j \cdot T_j \tag{2}$$

subject to

$$\sum_{b=1}^{n_b}\sum_{l=1}^{m} X_{jbl} = 1, \quad \forall j = 1,\ldots,n, \tag{3}$$

$$\sum_{j=1}^{n} X_{jbl} \le B, \quad \forall b = 1,\ldots,n_b, l = 1,\ldots,m, \tag{4}$$

$$\sum_{s=1}^{f} Y_{bls} = 1, \quad \forall b = 1,\ldots,n_b, l = 1,\ldots,m, \tag{5}$$

$$e_{js} \cdot X_{jbl} \le Y_{bls}, \quad \forall j = 1,\ldots,n, b = 1,\ldots,n_b, l = 1,\ldots,m, \tag{6}$$

$$p_s \cdot Y_{1ls} \le C_{1l}, \quad \forall l = 1,\ldots,m, s = 1,\ldots,f, \tag{7}$$

$$C_{(b-1)l} + \sum_{s=1}^{f} p_s \cdot Y_{bls} \le C_{bl}, \quad \forall b = 2,\ldots,n_b, l = 1,\ldots,m, \tag{8}$$

$$\left(C_{bl} - d_j\right) - M \cdot \left(1 - X_{jbl}\right) \le T_j, \quad \forall j = 1,\ldots,n, b = 1,\ldots,n_b, l = 1,\ldots,m, \tag{9}$$

$$C_{bl}, T_j \ge 0, X_{jbl}, Y_{bls} \in \{0,1\}. \tag{10}$$

The objective (2) intends to minimize TWT. Constraints (3) ensure that each job is assigned to a batch, and constraints (4) do not allow more than B jobs assigned to the same batch. With constraints (5) we make sure that each batch belongs to a single job family, and constraints (6) ensure that the families of the jobs assigned to a batch match the family of the batch. Using constraints (7) the completion times of the first batches on each machine are computed, whereas constraints (8) ensure the correct completion times for all subsequent batches. Finally, constraints (9) express the tardiness for each job. A similar formulation for minimizing the makespan can be found in [6].

Since the problem (1) is NP-hard, calculating the optimal solution of the above model for instances with more than 30 jobs is out of range for standard MIP solvers within a reasonable time.

2.2 Literature Review

Batching problems are considered by many researchers [4]. A survey of batching models in semiconductor manufacturing is provided in [8].

ACO is a population based metaheuristic designed to solve combinatorial optimization problems. ACO type approaches have been applied to different scheduling problems. Bauer *et al.* [2] developed first an ACO algorithm for the single machine total tardiness problem. ACO schemes are applied to problem $1||TWT$ in [3,9]. ACO type approaches were also suggested for scheduling problems related to parallel machines. Here, usually the assignment and the sequencing decisions are represented by pheromone trails. For example, the problem $P_m||TWT$ is studied using ACO in [13].

So far only some preliminary work related to batch scheduling with artificial ants is described in the literature. An ACO type approach to problem (1) is described in [12]. They use the Apparent Tardiness Cost (ATC) dispatching rule to form batches. ACO is used to assign batches to machines. However, we use a quite different representation that has the advantage to include batch formation, assignment, and sequencing. Li *et al.* [7] considered the scheduling problem $P_m \mid s_{kj}, r_j, incompatible, p-batch \mid TWT$ where r_j denotes the ready time of job *j*. They form batches using the time window technique suggested in [10] for batching problems. An AS type approach is used to assign the batches to machines and sequence them. However, only a few computational results are available. A quite different AS for the single machine scheduling problem $1 \mid p-batch, incompatible \mid \sum w_j C_j$ is suggested in [5].

3 Heuristic Approaches

3.1 ATC-BATC Heuristic

A simple list based scheduling approach is described for the sake of completeness. Every time a machine becomes free, one batch from each family is formed, and one of all the formed batches is selected to be scheduled next. The number of batches will be equal to the number of families that contain unscheduled jobs. The batch is chosen based on the ATC dispatching rule. This heuristic is simple but has been known to give reasonable solutions quickly [1]. The ATC index I_{js} for a job *j* belonging to family *s* calculated at time *t* is given below:

$$I_{js}(t) := w_{js} / p_s \, exp\left(- max\left(d_{js} - p_s - t, 0\right) / \kappa \bar{p}\right). \tag{11}$$

Here, we denote by t the time for decision-making, $\bar{p}$ is the average processing time of the remaining jobs, and κ is a look-ahead parameter. The jobs in each family are ordered in decreasing order of their ATC indices, and a batch of the first *B* jobs is formed per family. In order to schedule one of these batches the Batched Apparent Tardiness Cost (BATC) rule is used. Therefore, at time *t*, for all the batches the following index is calculated by $I_{BATC}(x,s,t) := \sum_{j \in B_{xs}} I_{js}(t)$, where $I_{BATC}(x,s,t)$ is the BATC index for batch *x* of family *s* at time *t*. The batch with the highest BATC index is scheduled. We call the heuristic that forms batches using ATC and sequence the batches by BATC in the remainder of this paper ATC-BATC-H. It is well known that ATC type rules lead to small TWT values when the look-ahead parameters are set appropriately [1]. We consider all $\kappa \in \{0.5, 1.0, \ldots, 5\}$, fix one of these values, solve (1) and take finally the value of κ that leads to the smallest TWT value for the given instance and ATC-BATC-H.

Furthermore, we will use a decomposition heuristic to improve the schedules obtained from ATC-BATC-H. From an initial sequence obtained by ATC-BATC-H a sub-sequence of size λ is solved into an optimal sequence (with respect to TWT) by using total enumeration. The first α batches of this optimal sequence are fixed into the final sequence, and the next λ batches are considered. This process is repeated until all batches are covered and no improvement is made on TWT. At maximum *iter* iterations are allowed. We call the heuristic based on the decomposition heuristic DH($\lambda, \alpha, iter$).

We use the swap procedure suggested in [1] for batching with parallel machines in order to improve the quality of the schedules. The swap procedure exchanges jobs for all batches of one family across the machines with the objective to reduce TWT. We denote the swap procedure in the remainder of this paper by Swap.

Note that in [1] batches are formed by using ATC-BATC-H. These batches are assigned to the machines via a GA. DH and Swap is used to improve the schedules. The schedules obtained by using these algorithms lead to the smallest TWT among all the tested heuristics. Therefore, we will use this heuristic for comparison purposes within this research. This approach is called ATC-GA1a-DH+swap in [1]. In this paper, we call it only GA for abbreviation. ATC-BATC-H is used as a reference heuristic within this paper.

3.2 ACS Scheme

We start with describing the used ACS approach. The scheduling problem (1) is represented as a pure sequencing problem denoted as

$$\pi := (\pi[1], \ldots, \pi[n]). \tag{12}$$

We construct a permutation by inserting the jobs of a newly formed and chosen batch to the partial sequence starting from an empty sequence and describe how we form appropriate batches.

Therefore, we consider a permutation of the jobs for each family *s* that are not included in the partial sequence of representation (12) $\pi_s^t := (\pi_s[1],\ldots,\pi_s[n_{s,curr}])$, where $\pi_s[i]$ denotes the job of family *s* that is sequenced on position *i* in π_s^t and the notation $n_{s,curr}$ is used for the number of jobs that are not already part of the partial schedule that corresponds to (12). The time *t* denotes the time where the newly formed batch can start with processing.

Starting from π_s^t, we obtain a sequence of batches by considering subsequences of the form $\chi_s(w) := (\pi_s[wB+1],\pi_s[wB+2],\ldots,\pi_s[(w+1)B])$, where $w \in \{0,\ldots,\lceil n_{s,curr}/B \rceil\}$. Note that only the last batch formed in this way may have less than *B* jobs. We assume that the batches are sequenced in decreasing order of importance, i.e., $\chi_s(w)$ is always more important than batch $\chi_s(w+1)$. The used measure of importance is related to the ACS approach and is described later. We select the most important batch among the $\chi_s(0), s = 1,\ldots,f$ and add it to the partial sequence to construct the job permutation (12).

We use a list based approach to assign the selected batch to the machines. Here, we simple use the machine that is available next to assign the next batch to it. This availability time is the *t* used in the notation π_s^t. By doing so, the job sequence (12) can be constructed to determine a solution of (1). Each solution of (1) can be transferred in a job sequence (12).

Based on this procedure, it is enough to describe a heuristic to find job permutations that lead to small values of TWT. We describe an ACO type approach to tackle this problem. The main idea of ACO consists in maintaining a set of artificial ants. Each ant starts with an empty sequence of the jobs and constructs a permutation by adding iteratively one of the remaining jobs to the partial sequence constructed so far.

The search for good sequences with respect to scheduling problem (1) is coupled between the different ants by artificial pheromone trails. The ants communicate indirectly by modifying the pheromone trails after the construction of a new permutation π. The solutions for (1) that correspond to the permutation can be further improved by local search techniques.

The construction of a solution corresponds to create a new ant in the general ACO scheme [3]. This ant takes the pheromone trail information into account while its construct a solution of problem (1). It updates the pheromone trail locally whenever it adds a new job to the job permutation (12). When the maximum number of ants per iteration is not reached we construct a new solution. Otherwise, we update pheromone trails globally by using the best solution obtained by the ants so far.

We have to describe the tailoring of the steps of the general ACO scheme to solve (1). We denote by η_{ij} the heuristic desirability for setting $\pi[i] := j$. The η_{ij} values are typically derived by using appropriate problem specific dispatching rules. In this research, we use the ATC dispatching rule with priority index (11). In this situation, *t* is the time when the next machine gets available.

We denote by $\tau_{ij}(l_{iter})$ the pheromone intensity that is associated with the setting $\pi[i] := j$, i.e., job *j* is placed on position *i* in π. The parameter l_{iter} is used for the current iteration of the ACO scheme. The maximum number of ants used within one iteration is denoted by $n_{a\,max}$. The construction of a solution within one iteration of the ACO scheme works as follows:

1. Initialize the number of the current ant by $n_a = 1$.
2. Create a new artificial ant. Denote the set of jobs that are not used to construct π by J_U. Initialize the set with $J_U := \{1,\ldots,n\}$. We introduce the set of unscheduled jobs $J_U^s := \{j \in J_U \mid j \text{ is part of family } s\}$ for each family *s*.

3. Create a sample of a $U[0,1]$ distributed random variable. Denote the obtained value by q_0. We assume that we have at least $n_{s,curr} > 0$ jobs in J_U^s. When $q_0 \le q$ then the $B_{s,curr} := min(B, n_{s,curr})$ jobs that maximize the value of

$$\Phi_{ij}^s := \frac{1}{B_{s,curr}} \sum_{g=0}^{B_{s,curr}-1} \left(\sum_{l=1}^{i+g} \tau_{lj+g}(l_{iter}) \right) \eta_{i+g\,j+g}^{\beta} \tag{13}$$

over all families s are set into the current partial sequence on position $i,\ldots,i+B_{s,curr}-1$. A similar pheromone summation rule is suggested in [9] for the scheduling problem $1||TWT$. Using the sum of pheromone values of job j up to position i offers the advantage that the selection of jobs j that are chosen by previous ants for positions smaller than i will be enforced. The quantity $\beta > 0$ is a parameter of ACS that is related to the importance of the desirability. Here, $q \in [0,1]$ is a fixed number that is a parameter of the ACS. Usually, q is selected close to 1. Hence, the ant makes the best decision with a probability q as indicated by the heuristic information and the pheromone trails. These decisions make sure an exploitation of the search space. When $q_0 > q$ then job $j \in J_U^s$ is selected according to the following discrete distribution with the probabilities p_{ij} given by

$$p_{ij} := \begin{cases} \dfrac{\left(\sum_{l=1}^{i} \tau_{lj}(l_{iter}) \right) \eta_{ij}^{\beta}}{\sum_{r \in J_U^s} \left(\sum_{l=1}^{i} \tau_{lr}(l_{iter}) \right) \eta_{ir}^{\beta}} & when \quad j \in J_U^s \\ 0 & otherwise. \end{cases} \tag{14}$$

For each family, we select a job according to this discrete probability distribution. We form the batch including this job and the next $B_{s,curr}-1$ jobs from a list that is sequenced in decent order according to the index $\left(\sum_{l=1}^{i} \tau_{lj}(l_{iter}) \right) \eta_{ij}^{\beta}$. Note that we choose only the first job of the batch according to (14) because we are not interested in completely destroying the structure of batches formed by ATC-BATC-H type rules. We evaluate the batches for all families according to expression (13) and consider the batch with the largest value Φ_{ij}^s. The ant performs a biased exploration with a probability of $1-q$.

4. A local step-by-step update of the pheromone trail is performed immediately after an ant has added all jobs of a batch to the partial sequence by

$$\tau_{ij}(l_{iter}) := (1-\theta)\tau_{ij}(l_{iter}) + \theta\tau_0, \tag{15}$$

where $\theta \in (0,1]$ and $\tau_0 > 0$ are parameters of the ACS. The local update is necessary to make the decision to put job j on position i less desirable for consecutive ants. The exploration of different sequences is enforced by performing the step-by-step update.

5. Set $J_U := J_U - B_{s,curr}$. When $J_U \neq \emptyset$ go to Step 3. When $J_U = \emptyset$ then improve the schedule by using DH and Swap. Change the position of the jobs in representation (12). Update the number of ants by $n_a := n_a + 1$. Go to Step 2 when $n_a \le n_{a\,max}$ is valid.

6. When job *j* is scheduled on position *i* in the global best solution at iteration l_{iter} then update the global pheromone trail by the following expression:

$$\tau_{ij}(l_{iter}+1) := (1-\rho)\tau_{ij}(l_{iter}) + \rho / TWT^{*}, \quad (16)$$

where $\rho \in (0,1]$ is a parameter of the ACO approach that is used to model the pheromone evaporation. TWT^{*} is the TWT value of either the global-best or the iteration-best solution. Note that also mixed strategies are possible.

7. Update the iteration number by $l_{iter} := l_{iter} + 1$. Start again with Step 1 when the termination condition is not fulfilled.

3.3 MMAS Scheme

We perform also some additional experimentation with a MMAS. Here, we set basically $q = 0$ in ACS and avoid any local update of the pheromone trails in Step 4 of the ACS scheme. In MMAS, the pheromone trail is bounded by an upper and a lower pheromone value (cf. [14] for a more detailed description of the MMAS principles). We denote these values by τ_{max} and τ_{min} respectively. We use the following settings for the two bounds according to [14] $\tau_{max} := 1/((1-\rho)TWT^{*})$, where TWT^{*} is the TWT value of the iteration-best solution, and $\tau_{min} := 2B\,\tau_{max}\left(1-\sqrt[n]{p_{best}}\right)/\left((n-2B)\sqrt[n]{p_{best}}\right)$ because an ant has to choose on average on $n/(2B)$ different jobs. Here, p_{best} is the probability that an ant constructs the best solution by *n* consecutive right decisions. When p_{best} is chosen too small than we may observe that $\tau_{min} > \tau_{max}$. In this case we have to set $\tau_{min} = \tau_{max}$. If p_{best} is close to one then τ_{min} is also close to zero. Hence, a careful choice of the parameter p_{best} is crucial for the performance of MMAS.

4. Computational Experiments

4.1 Design of Experiments and Implementation

In this section, we present results of computational experiments. We used test instances generated according to an extension of the test data description given in [1] to the case of more parallel machines. The test cases depend on the batch size, the number of parallel machines, the number of jobs of each job family, the due date range *R* and the tightness *T* of the schedules. In our experiments we consider the case of three different job families. Totally, we solve 480 different test instances.

All the tests are performed on an Intel Pentium D with 3.2 GHz CPU and 4 GB of random access memory with SUSE Linux 10 as operating system. The algorithms are implemented in the C++ programming language, for the GA implementation the framework Galib is used.

We also solved a couple of test instances for $f = 2, B = 4, m = 2, n = 16$ and for $f = 3, B = 4, m = 3, n = 24$ using the MIP (2)-(10) optimally. These experiments were performed with ILOG CPLEX 11.1. ACS found the optimum solution for all instances.

4.2 Comparision of ACS and GA

We perform experiments with four different amounts of time for computation in order to carry out a fair comparison of ACS and GA. The allowed time for computation is 5, 10, 30, and 60

seconds respectively. In order to select the various parameter of the ACO scheme, we performed extensive experiments on a subset of the generated test instances. Finally, we come up with $n_{a\,max} = 5$, $q = 0.9$, $\tau_0 := 1 / TWT_{best}$ where TWT_{best} denotes the best TWT value obtained by ATC-BATC-H. The global-best solution is used for global pheromone update with $\rho = 0.01$. The evaporation parameter for local update is $\theta = 0.1$. Furthermore, we set $\beta = 1$ within all experiments. DH is used for both ACS and GA. In order to compare the performance of the two solution techniques with GA, we simply apply the same parameter setting as [1], i.e., we set $\lambda = 5, \alpha = 2$ and $iter = 15$. The remaining parameters of the GA (except the stopping criteria) are the same as used in [1].

The results are presented in Table 1. We show the ratio of the TWT value of the metaheuristic obtained as average value from three independent runs and the TWT value of ATC-BATC-H. We present the time for computing immediately after the name of the heuristic.

Table 1: TWT Values Relative to ATC-BATC for Different Allowed Computing Times

Level	ACS5	ACS10	ACS30	ACS60	GA5	GA10	GA30	GA60
n = 180	0.9196	0.9171	0.9151	**0.9133**	0.9407	0.9294	0.9178	0.9171
n = 240	0.9213	0.9167	0.9122	**0.9101**	0.9521	0.9383	0.9195	0.9136
n = 300	0.9248	0.9212	0.9147	**0.9120**	0.9601	0.9475	0.9283	0.9195
B = 4	0.9255	0.9226	0.9191	**0.9166**	0.9615	0.9504	0.9324	0.9244
B = 8	0.9224	0.9181	0.9130	**0.9110**	0.9449	0.9307	0.9155	0.9132
m = 3	0.9285	0.9256	0.9222	**0.9210**	0.9556	0.9465	0.9353	0.9316
m = 4	0.9194	0.9156	0.9116	**0.9093**	0.9473	0.9373	0.9222	0.9177
m = 5	0.9180	0.9145	0.9097	**0.9073**	0.9491	0.9332	0.9149	0.9088
m = 6	0.9132	0.9091	0.9040	0.9011	0.9432	0.9280	0.9066	**0.9007**
R = 0.5 T = 0.3	0.7885	0.7800	0.7697	**0.7636**	0.8444	0.8148	0.7814	0.7740
R = 0.5 T = 0.6	0.9641	0.9620	0.9595	0.9582	0.9712	0.9659	0.9585	**0.9559**
R = 2.5 T = 0.3	0.9812	0.9797	0.9780	0.9771	0.9898	0.9848	0.9786	**0.9767**
R = 2.5 T = 0.6	0.9696	0.9673	0.9647	**0.9640**	1.0149	1.0043	0.9849	0.9764
Overall	**0.9259**	**0.9223**	**0.9180**	**0.9157**	**0.9551**	**0.9425**	**0.9258**	**0.9207**

Looking at the overall results the GA continuously improves with increasing runtime, but ACS reaches after 30 seconds already a better solution quality than the GA after twice that time. Furthermore, the results show that ACS convergence fast to a good solution and the additional solution quality gained by extending the runtime is small. For ACS the difference between 5 and 60 seconds runtime is less than 1%.

Looking at the detailed results for different number of jobs, respectively different number of families, it seems that the performance of the GA drops significantly if the problem gets more complex (more jobs). ACO show this performance difference only for short runtimes of 5 or 10 seconds, but those differences level out for longer runtimes.

Obviously the due dates (parameters *R* and *T*) have greatest impact on the solution quality. For test instances with a small range of the due dates (*R=0.5*) together with a low number of tardy jobs (*T=30%*) the ATC-BATC-H solution can be improved by more than 20%. If the range of the due dates is large (*R=2.5*) or the number of tardy jobs is high (*T=60%*) the average improvement of both methods is significantly less, but still the ordering GA > ACO with respect to solution quality is the same.

4.3 Computational Experiments with Different Pheromone Update Schemes and with MMAS

In this second series of experiments, we are interested in identifying the performance of MMAS and the impact of using global-best and iteration-best solution for globally updating the pheromone trails within ACS. In both situations, we do not prescribe a maximum amount of computing time. Instead, we allow for at maximum 1000 iterations. The parameter setting for MMACS is similar to ACS, except $\theta = 0$, $q = 0$, $\tau_0 = 10000.0$, and $\beta = 2$. We select $\beta = 5$ when the TWT value obtained by ATC-BATC-H is larger as 1000. The iteration-best solutions are chosen for pheromone trail updates. We also incorporated a pheromone trail smoothing as suggested in [14]. Note that we also performed for experiments with values ρ close to 1. However, we obtained for a majority of test instances better results by using the smaller value. In our experiments, $p_{best} = 0.005$ is chosen.

In Table 2, we show the TWT values for both MMAS and for ACS relative to the corresponding TWT values for ATC-BATC-H. We denote by ACS-M-GB-IB a mixed strategy that allows the global-best and the iteration-best solution to update the pheromone trails with an evaporation parameter $\rho = 0.01$. The abbreviation ACS-IB is used for ACS with iteration-best solution. The value of ρ is equals to 0.01 in this case. Finally, the scheme described in Section 3.2 is called ACS-GB.

Table 2: TWT Values for MMAS and Different Pheromone Update Schemes

Level	MMAS	ACS-M-GB-IB	ACS-IB	ACS-GB
$n = 180$	0.9134	0.9136	0.9120	0.9128
$n = 240$	0.9088	0.9092	0.9072	0.9082
$n = 300$	0.9100	0.9100	0.9094	0.9095
$B = 4$	0.9145	0.9156	0.9145	0.9147
$B = 8$	0.9109	0.9103	0.9086	0.9096
$m = 3$	0.9214	0.9208	0.9185	0.9196
$m = 4$	0.9087	0.9083	0.9072	0.9075
$m = 5$	0.9045	0.9057	0.9044	0.9052
$m = 6$	0.8999	0.9004	0.8996	0.8999
$R = 0.5, T = 0.3$	0.7543	0.7604	0.7589	0.7600
$R = 0.5, T = 0.6$	0.9571	0.9579	0.9576	0.9581
$R = 2.5, T = 0.3$	0.9815	0.9767	0.9755	0.9759
$R = 2.5, T = 0.6$	0.9658	0.9643	0.9619	0.9624
Overall	**0.9147**	**0.9148**	**0.9135**	**0.9141**

It turns out that differences between MMAS and ACS are small. But the computing time for MMAS is very often several minutes per instance. Furthermore, using the iteration-best solution for the global pheromone update provides only slightly better results. Among the four different strategies using the global-best and the iteration-best solution is the worst one.

5. Conclusions and Future Research

We suggested two ACO approaches for scheduling jobs with incompatible families on parallel identical batching machines. We compared the solution quality and the time needed for computation of the two ACO approaches with results for a GA suggested in previous research [1]. All the approaches were hybridized by a decomposition scheme and local search approaches.

It turned out that the two ACO approaches provide solutions with similar solution quality. The GA approach is slightly outperformed by ACS with respect to solution quality. Among the

three metaheuristics the ACS approach is the fastest one. A small time for computation is especially interesting when using one of the metaheuristic approaches within decomposition approaches like the shifting bottleneck heuristic for entire wafer fabs because in this case we have to solve a large number of scheduling problems for jobs on parallel machines [11].

Multiple order per job (MOJ) scheduling problems are important in 300 mm wafer fabs. Several customer orders have to be assigned to one Front-Opening-Unit-Pod (FOUP). The FOUPS travel around the wafer fab and the wafers of the FOUP will be processed on different machines. The problem of FOUP formation is similar to our batching problem. That is why we believe that we can use algorithms from this research to tackle MOJ scheduling problems.

Acknowledgements The authors gratefully acknowledge the testing efforts of Markus Erhardt.

References

1. Balasubramanian, H., Mönch, L., Fowler, J. W., Pfund, M., Genetic Algorithm Based Scheduling of Parallel Batch Machines with Incompatible Job Families to Minimize Total Weighted Tardiness, International Journal of Production Research, 42(8), 1621-1638, (2004)
2. Bauer, A., Bullnheimer, B., Hartl, R., Strauss, C., Minimizing Total Tardiness on a Single Machine Using Ant Colony Optimization, Central European Journal of Operations Research, 8(2), 125-141, (2000)
3. den Besten, M., Stützle, T., Dorigo, M., Ant Colony Optimization for the Total Weighted Tardiness Problem, Proceedings 6th International Conference Parallel Problem Solving from Nature (PPSN VI), 611-620, (2000)
4. Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M. Y., Potts, C. N., Tautenhahn, T., van de Velde, S., Scheduling a Batching Machine, Journal of Scheduling, 1, 31-54, (1998)
5. Kashan, A. H., Karimi, B., Scheduling a Single Batch Processing Machine with Arbitrary Job Sizes and Incompatible Job Families: an Ant Colony Framework, Journal of the Operational Research Society, 1-12, (2007)
6. Klemmt, A., Horn, S., Weigert, G., Hielscher, T. Simulation-based and Solver-based Optimization Approaches for Batch Processes in Semiconductor Manufacturing, Proceedings of the 2008 Winter Simulation Conference, 2041-2049, (2008)
7. Li, L., Qiao, F., Wu, Q. ACO-based Scheduling of Parallel Batch Processing Machines with Incompatible Job Families to Minimize Total Weighted Tardiness, Proceedings ANTS 2008, 219-228, (2008)
8. Mathirajan, M., Sivakumar, A. I., A Literature Review, Classification and Simple Meta-analysis on Scheduling of Batch Processors in Semiconductor, International Journal of Advanced Manufacturing Technology, 29, 990-1001, (2006)
9. Merkle, D., and Middendorf, M. An Ant Colony Optimization Algorithm with a New Pheromone Evaluation Rule for Total Tardiness Problems, Proceedings of the EvoWorkshops 2000, LNCS 1803, 287-296, (2000)
10. Mönch, L., Balasubramanian, H., Fowler, J. W., Pfund, M. E., Heuristic Scheduling of Jobs on Parallel Batch Machines with Incompatible Job Families and Unequal Ready Times, Computers & Operations Research 32, 2731-2750, (2005)
11. Mönch, L., Schabacker, R., Pabst, D., Fowler, J. W., Genetic Algorithm-based Subproblem Solution procedures for a Modified Shifting Bottleneck Heuristic for Complex Job Shops, European Journal of Operational Research, 177(3), 2100-2118, (2007)
12. Srinivasa Raghavan, N. R., Venkataramana, M., Scheduling Parallel Batch Processors with Incompatibe Job Families Using Ant Colony Optimization, Proceedings of the 2006 International Conference on Automation Science and Engineering, 507-512, (2006)
13. Srinivasa Raghavan, N. R., Venkataramana, M., Parallel Processor Scheduling for Minimizing Total Weighted Tardiness Using Ant Colony Optimization, Journal of Advanced Manufacturing Technology. To appear (2009)
14. Stützle, T., Hoos, H. H., MAX-MIN Ant System, Future Generation Computer Systems 16, 889-914, (2000)