Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

GENETIC ALGORITHMS TO SOLVE A SINGLE MACHINE MULTIPLE ORDERS PER JOB SCHEDULING PROBLEM

Oleh Sobeyko
Lars Mönch

Department of Mathematics and Computer Science
Universitätsstr.1
University of Hagen
Hagen, 58097, GERMANY

ABSTRACT

This research is motivated by a scheduling problem found in 300-mm semiconductor wafer fabrication facilities (wafer fabs). Front opening unified pods (FOUPs) are used to transfer wafers in wafer fabs. The number of FOUPs is kept limited because of the potential overload of the automated material handling system (AMHS). Different orders are grouped into one FOUP because orders of an individual customer very often fill only a portion of a FOUP. We study the case of lot processing and single item processing. The total weighted completion time objective is considered. In this paper, we propose a grouping genetic algorithm (GGA) to form the content of the FOUPs and sequence them. The GGA is hybridized with local search. Furthermore, we also study a random key genetic algorithm (RKGA) to sequence the orders and assign the orders to FOUPs by a heuristic. We compare the performance of the two GAs based on randomly generated problem instances with simple heuristics and other GAs from the literature. It turns out that GGA only slightly outperforms the previous genetic GAs but it is faster when a lot processing environment is considered. The RKGA behaves similar to the best performing GAs described in the literature with respect to solution quality and computing time.

1 INTRODUCTION

Semiconductor manufacturing is at the heart of the electronics industry. The wafer fabrication part of semiconductor manufacturing is very complex, consisting of hundreds of processing steps, a diverse product mix, re-entrant flows, sequence dependent set-ups, and batch processing, i.e., several lots can be processed at the same time on the same tool, and expensive tools in parallel (Gupta et al. 2006). An increased level of full-factory automation is typical for 300-mm wafer fabs. Therefore, an AMHS is very important for these modern wafer fabs (Agrawal and Heragu 2006 and Montoya-Torres 2006 for recent survey papers on AMHS in semiconductor manufacturing) mainly because it is difficult for human operators to carry the wafers with FOUPs.

A FOUP is a standard unit of lot transfer between tools. It is a container that holds up to 25-wafer lots of 300-mm wafers in an inert, nitrogen atmosphere. FOUPs are expensive. More importantly, a large number of FOUPs can cause a congested AMHS. Therefore, the number of FOUPs is limited and is definitely a restriction. Because of the combination of decreased line width and increased area per wafer fewer wafers are required to fill the integrated circuits (ICs) of a certain customer. Therefore, there is quite often a need to group orders of different customers into one FOUP. We call the resulting entity a job to be conform with scheduling literature. When such a grouping decision is made than the jobs have to be scheduled on the different tools within a wafer fab. Consequently, the resulting class of problems are called

multiple orders per job (MOJ) scheduling problems. Recently, problems of this type have been extensively studied by Mason and his group (cf. Mönch et al. 2009 for a survey of these problems). A static single machine MOJ scheduling problem with total weighted completion time (TWC) criterion is discussed by Mason et al. (2004). Mixed integer programming formulations and rather simple heuristics are proposed. Later, two genetic algorithms are suggested for the corresponding problem including ready times of the orders by Mason and Qu (2005). In this paper, we propose a GGA and a RKGA respectively for the problem researched by Mason et al. (2004). We show that the GGA slightly outperforms the two GAs from (Mason and Qu 2005), but it is much faster with respect to computing time in certain situations. The RKGA shows a similar behavior as the best performing GA described in the literature.

The paper is organized as follows. The researched problem is described in Section 2. We also provide some structural properties of the schedules. Related literature is discussed in Section 3. In Section 4, we propose the GGA and the RKGA. Finally, we present the results of computational experiments in Section 5.

2 PROBLEM STATEMENT AND ANALYSIS

We consider a single machine MOJ scheduling problem. We differentiate between lot and single item processing. Lot processing refers to the situation where the entire job, all wafers within the FOUP, will be processed simultaneously. The processing time is independent of the number of wafers within the FOUP. On the other hand, single item processing is related to the situation where the processing time of a job is the product of the wafer processing time and the number of wafers within the FOUP. Let p_j denote the processing time of job j. Jobs are labelled by $1,\ldots,J$. We have F FOUPs. We denote the size of order $o \in O$ by s_o. It is measured in number of wafers. We obtain $p_j = \rho \sum_{o \in j} s_o$, where ρ denotes the processing time of a single wafer in the single item processing case and $p_j = \rho$ for the lot processing environment. The set O is the set of all orders that have to be scheduled on the machine. We simply label the orders by $1,\ldots,N$. We assume that all orders have the same product type. Consequently, they can be grouped together. We assume that all FOUPs have the same capacity of K wafers and that $s_o \le K$. Orders cannot split into suborders. Clearly, we have $J \le \min\{F, N\}$.

Using the $\alpha \mid \beta \mid \gamma$ notation from scheduling theory (cf. Graham et al. 1979), the two different problems can be represented as

$$1 \mid moj(lot) \mid \sum w_o C_o \tag{1}$$

and

$$1 \mid moj(item) \mid \sum w_o C_o \tag{2}$$

respectively where we denote the completion time of order o by C_o and the weight of the order by w_o. The latter order attribute is used to express the importance of customer orders. The objective TWC is given by $\sum_{o=1}^{N} w_o C_o$. Two different decisions have to be made to solve the scheduling problems (1) and (2):

1. Job formation: Here the question is how to group the orders together into one job. Each single group corresponds to a FOUP.
2. Job Sequencing: After job formation, an appropriate sequence of the jobs has to be determined.

In order to find an optimal solution of problem (1) and (2) both decisions have to be made simultaneously. But because of the NP-hardness of problem (1) and (2) (see Mason and Chen 2009) often heuristics based on decomposition are reasonable. In the following, we assume that we are able to make the first de-

cision. It turns out that then the second decision is straightforward. We obtain the following two properties.

Property 1 When the job formation decision is made in the lot processing case, then the optimal job sequence can be determined by sorting the jobs with respect to non-increasing $\sum_{o \in j} w_o$ values.

Note that this property is a direct consequence of the fact that the weighted shortest processing time (WSPT) dispatching rule leads to an optimal schedule for the single machine scheduling problem with TWC objective (cf. Pinedo 2008). However, in the lot processing case, all the processing times are identical and therefore it is enough to consider the sum of the weights of the orders that form the FOUP.

Property 2 When the job formation decision is made in the item processing case, then the optimal job sequence can be determined by sorting the jobs with respect to non-increasing values of $\sum_{o \in j} w_o \Big/ \sum_{o \in j} s_o$.

The second property can be seen in a similar manner using the optimality properties of the WSPT dispatching rule. We will later use the two properties within our GAs.

3 PREVIOUS RELATED WORK

Heuristic approaches for solving the researched problem are described in (Mason et al. 2004). It turns out that two heuristics outperform the remaining ones in case of the lot processing and in case of the single item processing environment respectively. These heuristics are not iterative. Two different GAs are described in (Mason and Qu 2005). These heuristics are dedicated to the case when the orders have different ready times. It is interesting to note that scheduling is performed in two phases. In a first phase, the orders are sequenced and then the jobs are formed according to simple heuristics. However, both algorithms do not deal with the problem as a grouping problem.

Our research is based on the idea to treat the scheduling problem as a grouping problem. Based on Property 1 and 2, this is a reasonable approach, because the sequencing of the grouped entities is easy. Grouping problems can be solved efficiently by GGAs (Falkenauer 1998). Successful applications of GGA to bin packing problems (Falkenauer 1996) and pickup and delivery problems with time windows (Pankratz 2002) are known. To the best of our knowledge, only the paper by Singh et al. (2009) deal with the application of GGA type heuristics to machine scheduling problems so far.

4 SOLUTION APPROACHES

In this paper, two different GAs are proposed. We start with describing simple reference heuristics for benchmarking purposes. Then we present the GGA. Finally, we discuss the RKGA heuristic.

4.1 Simple Heuristics

The main idea of the heuristics proposed by Mason et al. (2004) is to decompose the scheduling problem into two phases. The first phase performs a sequencing of the orders. The orders are sequenced according to different criteria. The computational results show that the best results are obtained by the weighted smallest size (WSS) rule.

The second phase consists in forming jobs from the already sequenced orders. We apply different heuristics for lot and single item processing environments. In case of lot processing, the first-fit criterion FFD1 (cf. Mason et al. 2004) applied to the sequenced orders provides good results. This heuristic aims to fill the first jobs as much as possible. Thus, we obtain jobs with a large total weight. Jobs with larger weight correspond to smaller completion times. This results in a smaller TWC value.

In case of a single item processing environment, the FFD_AJS1 heuristic is applied (cf. Mason et al. 2004). The heuristic tries to use all of the available jobs in order to balance the total size of the orders within the single jobs.

The complexity of the heuristics is $O(N(\log N + F))$. It is a disadvantage of these simple heuristics that feasible solutions are not guarantied. Therefore, the heuristics require some additional effort to repair infeasible solutions.

4.2 GGA

Since all the orders have to be distributed between a limited amount of jobs, we have to deal with a grouping problem. Therefore, a GGA is assumed to be an appropriate method in this case. We propose two variants of the algorithm for both lot and single item processing mode. In order to apply a GGA, we have to look for an appropriate encoding scheme of solutions and for appropriate genetic operators. We start with solution encoding.

Every point of the search space of feasible solutions is represented as a sequence of jobs. Each job contains some subset of the orders. One order can be found only in one job. An example is shown in Figure 1. Here F_i is the i-th job in the processing sequence.

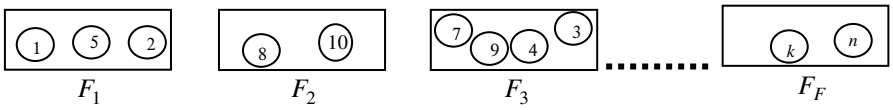

Figure 1: GGA encoding scheme

GGA is a population based algorithm. It requires a number of initial population elements, called genomes, that represent some part of the search space. We start with an initial population that is randomly generated. For genome initialization we generate an arbitrary permutation of the orders and then form the jobs by applying the assignment rules suggested by Mason et al. (2004).

We continue with describing the main genetic operators. The main purpose of crossover is combining parts of good feasible solutions to obtain better feasible solutions. A GA discovers the most promising parts of the search space moving towards the global optimum. Crossover has to act in such a way that each offspring inherits as much valuable information from its parents as possible. Indeed, if the parents contain good parts then these parts have to be preserved and inherited by children. For GGAs, the crossover operator has a specific form as it has to deal with entire groups rather than with separate objects. In our case, a set of orders from the same job are considered as one group. A specific crossover operator for GGA is proposed by Falkenauer (1998). Since a fixed amount of jobs can be used in the researched problem, this crossover is not appropriate for our problem because it allows a variable amount of jobs. Therefore, we propose another variant of grouping crossover that takes into account the maximum number of jobs. In this case, we directly create the child with an appropriate amount of jobs. The schema is shown in Figure 2. Here, F_{ij} is the j-th job of parent G_i. We denote the corresponding child by C.

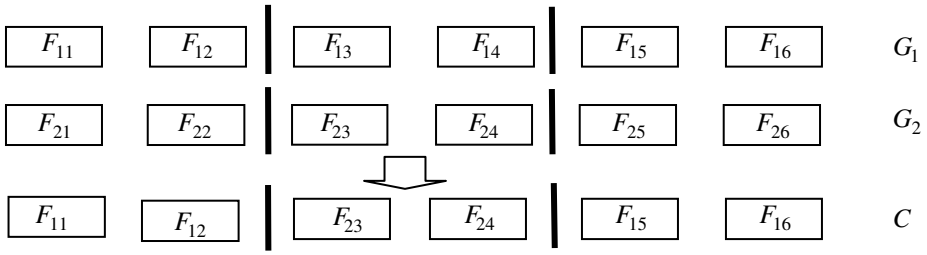

Figure 2: Grouping crossover

The crossover algorithm consists of the following steps:

1. Select randomly two crossing sites, delimiting the crossing section in each parent.
2. Replace the crossing section of parent G_1 with the crossing section of parent G_2.
3. Eliminate all orders now occurring twice from the job they were members of in the first parent. Thus, some jobs coming from the first parent have to be altered.
4. Adopt the resulting jobs according to the hard constraints and the objective function to optimize.
5. Repeat Step 2 till Step 4 with changed roles of the parents to obtain the second offspring.

Now the orders that are eliminated have to be reinserted into the child C again. As described in (Brown et al. 2003), the reinsertion strategy may have great influence on performance of the algorithm. Therefore, we apply the heuristics described in Subsection 4.1 as one of the best strategies of reinserting the uninserted orders. Sometimes it happens that not all of the uninserted orders can be reinserted. We repeat the crossover several times until a feasible offspring is obtained.

Having uninserted orders, we try to reinsert them in some local search manner in order to build better groups. For lot processing, we go through the list of jobs and try to replace for each of them up to three orders with one that is not inserted so far to increase the utilization and total weight of the jobs. More orders with smaller sizes become uninserted and it is easier to redistribute them later by applying the simple heuristic. In case of single item processing, for each job we try to reinsert up to three orders with one order in such a way that utilization of the job decreases but the total weight of it increases. Jobs with small utilization and high total weight are preferable because finally they are processed at the beginning of the sequence and provides smaller values of TWC. The remaining uninserted orders are reinserted by applying the simple heuristic.

Mutation is used to avoid a premature convergence of the GA towards a local optimum. We use a swap mutator as mutation operator. This means that in some genomes two jobs are selected randomly. In each of these two jobs we select randomly one order and then we exchange the two orders between the selected jobs preserving feasibility of the genome. Another possibility is exchanging more than two arbitrary orders in the selected jobs.

After crossover and mutation we improve the offspring using local search. The main idea of the local search is to swap orders across jobs to improve fitness of the genome. This technique depends on the type of processing. In case of lot processing, we exchange orders between any pair of jobs in order to obtain improvement of TWC. Let job i be processed before job j. The procedure works as follows. We select each order o_i from job F_i and o_j from F_j respectively. Then we check whether it is possible to exchange them preserving feasibility of the genome. If yes then in case o_j has a larger weight than o_i the orders are exchanged. We do not have to recalculate the objective function of the genome in order to verify whether the genome is better or not after the exchange. The used local search procedure for lot processing is shown in Figure 3.

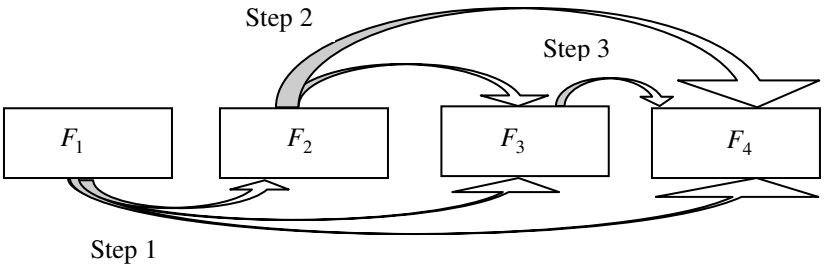

Figure 3: Local search steps in a lot processing environment

Firstly, we try to exchange orders between jobs F_1 and F_2 then between F_1 and F_3, and so on. After that we repeat this procedure between jobs F_2 and F_3, F_2 and F_4, and so on.

We use a similar exchange procedure in case of single item processing. We can easily verify whether the exchange improves the genome if the exchange is performed between two adjacent jobs F_i and F_{i+1}. Suppose we consider two adjacent jobs F_i and F_{i+1}. We select an order o_i from F_i and an order o_{i+1} from F_{i+1}. Let s_i and s_{i+1} be the sizes and w_i and w_{i+1} the weights of o_i and o_{i+1} respectively. Denote the total weight of job F_i by W_i and the total size of the orders in F_{i+1} by S_{i+1} before the exchange. We can prove that if the condition

$$(w_{i+1} - w_i)(S_{i+1} - s_{i+1}) + (W_i - w_i)(s_i - s_{i+1}) + w_{i+1}\, s_i - w_i\, s_{i+1} > 0 \tag{3}$$

is satisfied then swapping orders o_i and o_{i+1}, if it is possible, leads to a smaller value of TWC and, therefore, to better fitness of the genome. The described procedure of order exchanges is depicted in Figure 4.

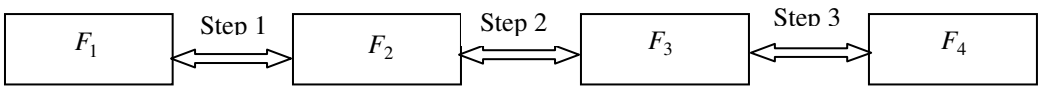

Figure 4: Local search steps in a single item processing environment

We start to exchange orders between the jobs F_1 and F_2, then between F_2 and F_3, and so on until all the jobs are covered. After one such pass there might still be potential for further improvement. Therefore, we apply several passes until a further improvement is not possible. The procedure is motivated as follows. Even if some order o is placed into the first job F_1 in the optimal solution, and we are dealing with a non-optimal solution, then after several passes the order can be shifted from the last job to the first job. Applying a local search technique leads to larger computational efforts within each iteration of the algorithm. But the performance of GGA is better. Hence, the GGA converges faster towards a solution with a small TWC value.

Besides crossover and mutation we also apply inversion techniques in order to bring together the most promising jobs in the genome. Thus every time before recalculation of the TWC value of a genome, we sort the jobs by applying the two properties shown in Section 2. Thus, we obtain the optimal sequence of the jobs.

4.3 RKGA

Unlike the GGA the RKGA approach (see Bean 1994) does not deal directly with groups but rather with sequences of orders which later are distributed between jobs according to some heuristic distribution rule as described in Subsection 4.1. We propose two variants of the algorithm for lot and single item processing.

In this case every point of the search space of solutions is encoded as a sequence of real numbers $z_i \sim U[0,1]$, called random keys. According to the random keys all the orders can be sequenced. We point out that not each such random key sequence represents a feasible solution because for some sequences not all orders can be distributed. An example of a genome and its decoding can be seen in Figure 5.

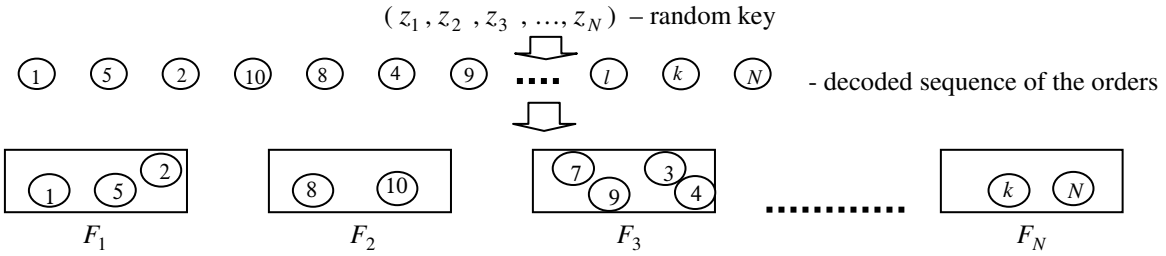

Figure 5: Genome decoding for RKGA

Having the random keys, we can decode a sequence of the orders before the redistribution. We assume that order i is sequenced for scheduling before order j only if $z_i < z_j$. Like the GGA the RKGA requires an initial population. The population is initialized as described for GGA.

The crossover operator acts on two genomes. We use a standard two-point crossover. The crossover works as follows. We randomly select two crossing sites for the parent genomes. Then we exchange the content of the crossing sections between the parents. Here we work with the random keys and not directly with the sequence of orders. The procedure is shown in Figure 6.

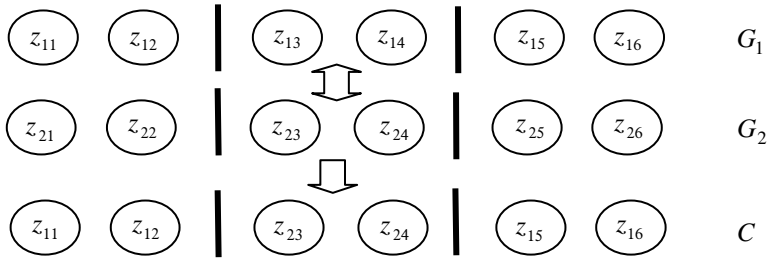

Figure 6: Random key two-point crossover

The second offspring is generated in a similar manner by exchanging the roles of the parents. Now the child genomes C_1 and C_2 contain random keys which allows us to decode the sequence of the orders before the distribution process. Although after each crossover we obtain a feasible sequence of orders by decoding the random keys, the genome may not be feasible. The problem that not all orders can be distributed can arise when using the FFD1-rule for lot processing or the FFD_AJS1-rule for single item processing. Such infeasible offspring have to be avoided. Therefore, we apply the crossover procedure several times until the obtained offspring is feasible. The mutation is implemented as a random swap of two different elements of the random key representation.

Similar local search techniques are used as described for GGA. In contrast to GGA, we run the RKGA without changing the random keys after the local search because there is no direct counterpart in the random key representation. The local search is performed before we calculate TWC. In this case, potentially the most promising genomes will survive. The same inversion as for GGA is applied for RKGA each time before we calculate the TWC value of a genome to determine its fitness. Again, this leads to a speed-up of the algorithm.

5 COMPUTATIONAL EXPERIMENTS

We start with describing our design of experiments. Then we present the computational results based on the randomly generated problem instances from Subsection 5.1. The simple heuristics, GGA, RKGA, and MGA, the best performing GA from (Mason and Qu 2005), are compared.

5.1 Design of Experiments

We generate randomly problem instances that depend on the number of available jobs, the number of orders, the size, and weight of the orders according to the generation scheme found in (Mason et al. 2004). The parameter ν defines the range of the order size, whereas the parameter β is responsible for the capacity of the FOUPs. The used design of the experiments is presented in Table 1.

We apply a steady-state GA with overlapping populations and a replacement rate of 0.5. The crossover probability is selected as 0.8. The corresponding mutation probability is 0.1. The population size is 300. Two hundred iterations per problem instance are performed by the GGA and one thousand iterations by RKGA and by MGA. These parameter values are chosen after extensive computational experiments based on a trial and error strategy. The three GAs are implemented in the programming language C++ us-

ing the framework GALib. A computer with Intel Core 2 Duo 2.26 MHz processor and 4 GB of main memory is used to carry out the experiments.

5.2 Computational Results

We compare the performance of the proposed heuristics by applying them to the problem instances. Since the researched problem is NP-hard, we are not always able to find optimal solutions within a reasonable amount of time when the number of orders and jobs is large. Thus, we start with assessing the proposed algorithms for small-sized problem instances using only one instance per factor combination. The considered number of orders in the small-sized problem instances is 10, 15, 20, and 25.

Table 1: Design of experiments

Factor	Level	Count
s_o (in wafer)	$\sim DU\left[v-\frac{v+1}{2}, v+\frac{v+1}{2}\right]$, $v \in \{3,5\}$	2
w_o	$\sim DU[1,15]$	1
K (in wafer)	$12\beta+1$, $\beta \in \{1,2\}$	2
N	*10,15,30,50,100*	5
F	$\lceil Nv/12\beta \rceil+1$	1
P (problem type)	single item, lot processing, $\rho=10$ in both cases	2
Number of independent problem instances	5 per factor combination	
Total number of problem instances		200

Table 2 presents the ratios of the TWC values for the various heuristics and the TWC values of the simple heuristics (SH). As it can be clearly seen, we obtain up to 4.5% better results in the lot processing case and up to 12% improvement in the single item processing case with respect to SH.

Table 2: Relative TWC values for small-sized problem instances

			Lot processing					Single item processing				
N	β	v	GGA	RKGA	MGA	SH	CPLEX	GGA	RKGA	MGA	SH	CPLEX*
10	1	3	1.000	1.000	1.000	1.000	1.000	0.946	0.946	0.946	1.000	0.946
	1	5	0.996	0.996	0.996	1.000	0.996	0.951	0.951	0.959	1.000	0.951
	2	3	1.000	1.000	1.000	1.000	1.000	0.872	0.872	0.878	1.000	0.872
	2	5	1.000	1.000	1.000	1.000	1.000	0.965	0.965	0.968	1.000	0.965
15	1	3	1.000	1.000	1.000	1.000	1.000	0.960	0.960	0.960	1.000	0.960
	1	5	0.955	0.955	0.955	1.000	0.955	0.970	0.970	0.970	1.000	0.970
	2	3	0.983	0.983	0.983	1.000	0.983	0.923	0.923	0.943	1.000	0.923
	2	5	0.987	0.987	0.987	1.000	0.987	0.927	0.927	0.930	1.000	0.927
20	1	3	0.997	0.997	0.997	1.000	0.997	0.940	0.940	0.940	1.000	0.940
	1	5	-	-	-	Failed	-	0.975	0.975	0.976	1.000	0.981 *
	2	3	1.000	1.000	1.000	1.000	1.000	0.879	0.879	0.905	1.000	0.879
	2	5	0.996	0.996	0.996	1.000	0.996	0.953	0.953	0.953	1.000	0.957 *
25	1	3	1.000	1.000	1.000	1.000	1.000	0.960	0.960	0.96	1.000	0.960 *
	1	5	0.957	0.957	0.96	1.000	0.957	0.967	0.967	0.967	1.000	0.987 *
	2	3	1.000	1.000	1.000	1.000	1.000	0.959	0.959	0.961	1.000	0.959 *
	2	5	0.993	0.993	0.993	1.000	0.993	0.942	0.943	0.942	1.000	0.946 *
Total			0.987	0.987	0.988	1.000	0.987	0.948	0.948	0.951	1.000	0.941 *

We also try to solve the mixed integer programs from (Mason et al. 2004) for our problem instances with CPLEX in order to assess the quality of our heuristic solutions. Because of the hardness of the researched scheduling problems, we were not able to find the optimal solutions with CPLEX within 4 hours per instance in some situations. Thus, the best found feasible solutions are marked with "*" in this situation. It is interesting to note that GGA always provides the optimal solution for each problem instance for which we are able to find this solution with CPLEX.

MGA does not always find the optimum but provides near-to-optimal solutions at the same time. RKGA can compete with these algorithms. We notice that SH is not able to find a feasible solution for one problem instance.

Next, we present the computational results for large-sized problem instances as well. Here, we generate five problem instances per factor combination and show the average values for the performance measures. The problem instances are generated with respect to solvability by the SH. In this situation, the GAs are terminated after two minutes.

Table 3: Relative average and best TWC values for five problem instances per factor combination

			Lot processing				Best	Single item processing				Best
N	β	ν	GGA	RKGA	MGA	SH	GGA	GGA	RKGA	MGA	SH	GGA
10	1	3	0.981	0.981	0.981	1.000	0.970	0.929	0.929	0.929	1.000	0.884
	1	5	0.987	0.987	0.987	1.000	0.969	0.953	0.953	0.953	1.000	0.891
	2	3	0.997	0.997	0.997	1.000	0.987	0.92	0.920	0.936	1.000	0.865
	2	5	0.985	0.985	0.985	1.000	0.965	0.927	0.927	0.929	1.000	0.845
15	1	3	0.988	0.988	0.988	1.000	0.948	0.931	0.931	0.931	1.000	0.892
	1	5	0.962	0.962	0.962	1.000	0.919	0.981	0.981	0.985	1.000	0.966
	2	3	1.000	1.000	1.000	1.000	1.000	0.943	0.943	0.945	1.000	0.919
	2	5	0.980	0.980	0.980	1.000	0.964	0.936	0.936	0.941	1.000	0.921
30	1	3	0.996	0.996	0.996	1.000	0.991	0.961	0.963	0.963	1.000	0.945
	1	5	0.970	0.970	0.971	1.000	0.932	0.978	0.980	0.978	1.000	0.955
	2	3	1.001	0.999	0.999	1.000	0.997	0.946	0.946	0.955	1.000	0.918
	2	5	0.977	0.977	0.979	1.000	0.952	0.97	0.971	0.971	1.000	0.963
50	1	3	0.997	0.998	0.999	1.000	0.995	0.979	0.989	0.981	1.000	0.972
	1	5	0.968	0.97	0.972	1.000	0.959	0.988	1.000	0.992	1.000	0.98
	2	3	0.996	0.996	0.999	1.000	0.988	0.959	0.961	0.966	1.000	0.951
	2	5	0.985	0.987	0.986	1.000	0.977	0.983	0.992	0.984	1.000	0.978
100	1	3	0.996	1.005	1.001	1.000	0.99	0.994	1.067	0.996	1.000	0.99
	1	5	0.975	0.981	0.991	1.000	0.966	0.995	1.025	1.005	1.000	0.992
	2	3	1.000	1.010	1.003	1.000	0.998	0.988	1.020	0.991	1.000	0.982
	2	5	0.985	0.994	1.002	1.000	0.979	0.994	1.060	1.000	1.000	0.99
Total			0.984	0.9900	0.993	1.000	0.972	0.987	1.023	0.992	1.000	0.944

From Table 3 we can see that GGA always provides results of a slightly better quality, especially for relatively small-sized problem instances. But when the size of the problem instances grow, the performance of the GAs decreases compared to the simple heuristics. We assume that this effect is caused by the increasing amount of small orders. Thus, the orders can be distributed more easily between the jobs and therefore, such problem instances are easier to solve by the SHs.

We also point out that the GGA slightly outperforms MGA. Of course, the improvement compared to the simple heuristics depends on the problem instance. Therefore, we analyze not only the average improvement but also the results for every separate problem instance. The best relative values found by the GGA are also presented in Table 3. We see that for small problem instances the improvement can be quite large.

Our computational experiments show that GGA outperforms the other GAs with respect to the number of iterations. Although one iteration of the GGA is more time consuming, it provides a larger improvement of the objective function value.

In case of lot processing the GGA has quite good performance when the time of computation is limited. We consider different maximum computing time levels for out experiments. The numerous computational experiments show that already after 10 seconds of computation per problem instance for large-sized problem instances with an amount of 50 and 100 orders the quality of solutions of the GGA is larger than the quality of solutions obtained by the MGA and the SHs.

For the case of a single item processing environment, GGA is outperformed by MGA when only a small amount of computing time is allowed. This is caused by the relatively large computational effort of one iteration of the GGA and sometimes infeasible child genomes after the crossover. But as the number of iterations increases the solutions improve. Finally, GGA also slightly outperforms MGA when enough computing time is available.

6 CONCLUSIONS AND FUTURE RESEARCH

In this paper, we discussed heuristics that are motivated by scheduling problems that can be found in 300-mm wafer fabs. Lots are transported using FOUPs. It appears that often orders have to be grouped together to fill a single FOUP. We proposed two different GAs. The first algorithm is a GGA as proposed by Falkenauer (1998). The second algorithm starts from order sequences and transforms each of the sequences into a sequence of jobs. The order sequences are determined by a RKGA. It turned out that the GGA only slightly outperforms GAs from previous research. But GGA is much faster when a lot processing environment is assumed. RKGA behaves similar to the best performing GA from previous research. We conclude that GGA is appropriate to solve MOJ scheduling problems because of its group-centric point of view.

There are several directions for future research. First of all, we are interested in studying problems with ready times, i.e., it is not required that all orders are available at the beginning, and due dates for the orders. A first preliminary experimentation shows that the GGA clearly outperforms the MGA in this situation. Furthermore, we are also interested in extending the GGA approach to the case of parallel machines.

REFERENCES

Agrawal, G. K., and S. S. Heragu. 2006. A Survey of automated material handling systems in 300-mm semiconductor fabs. *IEEE Transactions on Semiconductor Manufacturing* 19(1):112-120.

Brown E., and R. Sumichrast. 2003. Impact of the replacement heuristic in a grouping genetic algorithm. *Computers & Operations Research* 30:1575-1593.

Bean, J. C. 1994. Genetic algorithms and random keys for sequencing and optimization. *ORSA. Journal on Computing*, 6:154–160.

Falkenauer, E. 1998. *Genetic Algorithms and Grouping Problems*. *Wiley*.

Falkenauer, E. 1996. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics* 2:5-30.

Gupta, J. N., D. R. Ruiz, J. W. Fowler, and S. J. Mason. 2006. Operational planning and control of semiconductor wafer production. *Production Planning & Control* 17(7):639-647.

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

Mason, S., and J.-S. Chen. 2009. Scheduling multiple orders per job in a single machine to minimize total completion time. *Submitted for publication.*

Mason S., P. Qu, E. Kutanoglu, and J. Fowler. 2004. The single machine multiple orders per job scheduling problem. *Submitted for publication.*

Mason S., and P. Qu. 2005. Metaheuristic scheduling of 300-mm lots containing multiple orders. *IEEE Transactions on Semiconductor Manufacturing* 18:633-643.

Mönch, L., J. W. Fowler, S. Dauzère-Pérès, S. J. Mason, and O. Rose. 2009. Scheduling semiconductor manufacturing operations: problems, solution techniques, and future challenges. *Working Paper ENSM-SE CMP WP 2009/16*.

Montoya-Torres, J. R. 2006. A literature survey on the design approaches and operational issues of automated wafer-transport systems for wafer fabs. *Production Plannng & Control*, 17(6):648-663.

Pankratz, G. 2005. A grouping genetic algorithm for the pickup and delivery problem with time windows. *OR Spectrum* 27(1): 21-41.

Pinedo, M. 2008. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, Third Edition.

Singh, A., M. Sevaux, and A. Rossi. 2000. A hybrid grouping genetic algorithm for multiprocessor scheduling. In *Proceedings of IC3, CCIS 40*, 1-7.

AUTHOR BIOGRAPHIES

OLEH SOBEYKO is a PhD student and research and teaching assistant in the Department of Mathematics and Computer Science at the University of Hagen, Germany. He received a master's degree in applied mathematics from the National Ivan Franko University of Lviv, Ukraine. His current research interests are in applied optimization applications in manufacturing, logistics and service operations, and in multi-agent systems. His email address is <Oleh.Sobeyko@fernuni-hagen.de>.

LARS MÖNCH is Professor in the Department of Mathematics and Computer Science at the University of Hagen, Germany. He received a master's degree in applied mathematics and a Ph.D. in the same subject from the University of Göttingen, Germany. His current research interests are in simulation-based production control of semiconductor wafer fabrication facilities, applied optimization and artificial intelligence applications in manufacturing, logistics, and service operations. He is a member of GI (German Chapter of the ACM), GOR (German Operations Research Society), SCS, INFORMS, and IIE. His email address is <Lars.Moench@fernuni-hagen.de>.